

Air Force Institute of Technology

AFIT Scholar

Theses and Dissertations

Student Graduate Works

3-2002

Portable High-Performance Indexing for Vector Product Format Spatial Databases

Engin Colak

Follow this and additional works at: <https://scholar.afit.edu/etd>



Part of the [Graphics and Human Computer Interfaces Commons](#)

Recommended Citation

Colak, Engin, "Portable High-Performance Indexing for Vector Product Format Spatial Databases" (2002).
Theses and Dissertations. 4406.
<https://scholar.afit.edu/etd/4406>

This Thesis is brought to you for free and open access by the Student Graduate Works at AFIT Scholar. It has been accepted for inclusion in Theses and Dissertations by an authorized administrator of AFIT Scholar. For more information, please contact AFIT.ENWL.Repository@us.af.mil.



**PORTABLE HIGH-PERFORMANCE INDEXING FOR
VECTOR PRODUCT FORMAT SPATIAL DATABASES**

THESIS

Engin Colak, Lieutenant, TUAF

AFIT/GCE/ENG/02M-02

DEPARTMENT OF THE AIR FORCE

AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

Report Documentation Page

Report Date 8 Mar 02	Report Type Final	Dates Covered (from... to) Mar 01 - Mar 02
Title and Subtitle Portable High-Performance Indexing for Vector Product Format Spatial Databases	Contract Number	
	Grant Number	
	Program Element Number	
Author(s) Lt Engin Colak, TUAF	Project Number	
	Task Number	
	Work Unit Number	
Performing Organization Name(s) and Address(es) Air Force Institute of Technology Graduate School of Engineering and Management (AFIT/EN) 2950 P Street, Bldg 640 WPAFB OH 45433-7765	Performing Organization Report Number AFIT/GCE/ENG/02M-02	
Sponsoring/Monitoring Agency Name(s) and Address(es) AFRL/SNZW ATTN: Mike R. Foster 2241 Avionics Circle, Bldg 620,S1D34 WPAFB OH 45433-7303	Sponsor/Monitor's Acronym(s)	
	Sponsor/Monitor's Report Number(s)	
Distribution/Availability Statement Approved for public release, distribution unlimited		
Supplementary Notes The original document contains color images.		
Abstract Geo-spatial databases have an overall performance problem because of their complexity and large size. For this reason, many researchers seek new ways to improve the overall performance of geo-spatial databases. Typically, these research efforts are focused on complex indexing structures and query processing methods to capture the relationships between the individual features of fully-functional geo-spatial databases. Visualization applications, such as combat simulators and mission planning tools, suffer from the general performance problems associated with geo-spatial databases. This research focuses on building a high-performance geo-spatial database for visualization applications. The main approach is to simplify the complex data model and to index it with high-performance indexing structures. Complex features are reduced to simple primitives, then indexed using a combination of a disk-based array and B+-Trees. Test results show that there is a significant performance improvement gained by the new data model and indexing schema for low to medium zoom levels. For high zoom levels, there is a performance drop due to the indexing schemas overhead.		

Subject Terms Geo-spatial Databases, Geo-spatial Indexing, Geographic Information Systems, Vector Product Format.	
Report Classification unclassified	Classification of this page unclassified
Classification of Abstract unclassified	Limitation of Abstract UU
Number of Pages 123	

The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or United States Government.

Portable High-Performance Indexing For Vector Product Format

Spatial Databases

Presented to the faculty of the Graduate School of Engineering & Management

of the Air Force Institute of Technology

Air University

In Partial Fulfillment of the

Requirements for the Degree of

Master of Science (Computer Engineering)

Engin Colak, B. S.

Lieutenant, TUAF

March 2002

Approved for public release, distribution unlimited

ACKNOWLEDGMENTS

I would like to express my appreciation to my advisor Maj. Karl S. Mathias for his encouragement, expert guidance, and help in my research to explore and develop unfamiliar concepts and ideas. I also owe much appreciation to those who have contributed to this work.

Above all, I would like to express my admiration and gratitude to my lovely wife. She has never complained about being alone as an AFIT spouse in a foreign country, even while she was taking care of our wonderful child. I also would like to express my love to our lovely child for every simple thing he does which brings me to real life and reminds me of the value of my family, when I was struggling. My wife's contribution and our child's smiles helped me make it through our AFIT experience.

Engin Colak

TABLE OF CONTENTS

ACKNOWLEDGMENTS	IV
TABLE OF CONTENTS	V
LIST OF FIGURES	IX
LIST OF TABLES	XI
ABSTRACT.....	XII
I. INTRODUCTION.....	1
1.1 Definition of Terms.....	2
1.2 Background.....	3
1.3 Problem Statement	5
1.4 Objectives	6
1.5 Problem Scope	7
1.6 Standards.....	8
1.7 Research Focus	8
1.8 Summary.....	10
II. LITERATURE REVIEW	11
2.1 Introduction.....	11
2.2 Spatial Databases	12
2.2.1 Geographic Information Systems (GIS).....	12
2.2.2 The Open GIS Consortium (OGC).....	13
2.2.3 Properties of Spatial Database Management Systems.....	13
2.2.4 Traditional Databases Versus Spatial Databases.....	15
2.2.5 Data Model	16

2.2.6 Query Processing	18
2.2.6.1 Query Language	22
2.2.6.2 Other Accomplishments and Research on Query Processing	22
2.2.7 Indexing	23
2.2.8 Commercial Systems	28
2.3 Vector Product Format	28
2.3.1 VPF Characteristics	29
2.3.2 VPF Data Model	30
2.3.2.1 Data Organization.....	30
2.3.2.1.1 Directory	30
2.3.2.1.2 Tables.....	31
2.3.2.1.3 Indexes	31
2.3.2.2 Data Model Components	32
2.3.2.2.1 Feature Classes.....	33
2.3.2.2.2 Coverage	36
2.3.2.2.3 Library	39
2.3.2.2.4 Database.....	40
2.3.2.3 Data Quality	40
2.4 Visualizations.....	40
2.4.1 ArcGIS.....	41
2.4.2 VPFView	41
2.4.3 OpenMap	42
2.4.3.1 MapBean	43
2.4.3.2 Layers	44
2.4.3.3 Projection	45
2.5 Summary	45
III. METHODOLOGY AND DESIGN	46
3.1 Introduction.....	46
3.2 General Approach	47

3.2.1 Data Model	50
3.2.2 Indexing	52
3.2.3 Conversion	53
3.2.4 Visualization	54
3.3 Evaluation	54
3.4 Summary	55
IV. IMPLEMENTATION	56
4.1 Introduction.....	56
4.2 Proposed Data Model.....	56
4.2.1 Visibility Problem.....	60
4.2.2 Solving Visibility Problem.....	61
4.3 Indexing Schema.....	64
4.4 Improving The Data Model and Indexing Schema	68
4.5 Conversion	79
4.5.1 Data Preparation	79
4.5.2 Building The New Data Model and Indexing Schema	81
4.5.3 GUI	82
4.6 Visualization	85
4.6.1 GeoSVis Layer.....	85
4.6.2 Layer Graphic Warehouse	87
4.6.3 GUI	88
4.7 Summary	90
V. RESULTS	92
5.1 Introduction.....	92
5.2 Testing Method	92
5.3 Test Patterns.....	93
5.3.1 Pattern One	94
5.3.2 Pattern Two.....	96

5.3.3 Pattern Three.....	97
5.3.4 Pattern Four.....	98
5.3.5 Pattern Five.....	98
5.4 Conclusions.....	99
VI. CONCLUSIONS AND FUTURE WORK.....	102
6.1 Conclusion	102
6.2 Future Work.....	103
6.2.1 Data Model and Indexing Schema	103
6.2.2 Conversion.....	103
6.2.3 GeoSVis Layer.....	103
6.3 Summary.....	104
BIBLIOGRAPHY	106
VITA	109

LIST OF FIGURES

FIGURE 1. THE DATA MODEL OF THE OGC SPECIFICATION (CATTENSTART, 1999).	17
FIGURE 2. APPROXIMATIONS BY MINIMUM BOUNDING RECTANGLE/BOX.	19
FIGURE 3. N-CORNER CONVEX POLYGON APPROXIMATION.	20
FIGURE 4. USING APPROXIMATIONS AT FILTER AND REFINE TECHNIQUES.....	21
FIGURE 5. A SIMPLE QUERY TREE FOR A SPATIAL DATABASE (SHEKHAR, 1999).....	23
FIGURE 6. CURVE FILLING METHODS (SHEKHAR, 1999).	25
FIGURE 7. MAPPING FEATURES USING AN R-TREE INDEXING STRUCTURE (BERTINO, 1999).....	27
FIGURE 8. VPF DATA MODEL LEVELS (DoD, 1996).	33
FIGURE 9. GEOGRAPHIC PRIMITIVES IN VPF (DoD, 1996).	34
FIGURE 10. FEATURE CLASS STRUCTURAL SCHEMA (DoD, 1996).	36
FIGURE 11. COVERAGE CLASS STRUCTURAL SCHEMA (DoD, 1996).	37
FIGURE 12. VPF TOPOLOGICAL LEVELS (DoD, 1996).	38
FIGURE 13. TILED AND UNTILED COVERAGE (DoD, 1996).	39
FIGURE 14. OVERALL OPENMAP ARCHITECTURE (BBNT, 2001).	43
FIGURE 15. DETAILED OPENMAP ARCHITECTURE (BBNT, 2001).	43
FIGURE 16. ORGANIZING LAYERS VIA MAPBEAN (BBNT, 2001).	44
FIGURE 17. EDGE PRIMITIVE AND ITS RELATIONS WITH OTHER PRIMITIVES.	49
FIGURE 18. QUERYING SPATIAL DATA VIA APPROXIMATIONS.....	51
FIGURE 19. ELIMINATING COMPLEX FEATURES.	51
FIGURE 20. NEW INDEXING SCHEMA.....	53
FIGURE 21. PROPOSED DATA MODEL.	57
FIGURE 22. FILE HIERARCHY.....	59
FIGURE 23. REPRESENTATION OF DATA MODEL.	60
FIGURE 24. VISIBILITY PROBLEM.	61

FIGURE 25. SOLVING VISIBILITY PROBLEM.	64
FIGURE 26. PROPOSED INDEXING STRUCTURE.	66
FIGURE 27. FILE HIERARCHY.	67
FIGURE 28. FEATURE DATA AND METADATA FILE STRUCTURES.	68
FIGURE 29. MAPPING TWO DIMENSIONAL KEY INTO ONE DIMENSION.	70
FIGURE 30. FORCING A B-TREE FOR A DEPTH OF TWO.	71
FIGURE 31. LINKING CELLS VERTICALLY TOWARDS SOUTHEAST.	73
FIGURE 32. LINKING CELLS VERTICALLY TOWARDS SOUTHWEST.	73
FIGURE 33. FINAL INDEXING SCHEMA.	74
FIGURE 34. SOLVING EMPTY CELL PROBLEM FOR THE FINAL INDEXING STRUCTURE.	75
FIGURE 35. A QUERY EXAMPLE.	77
FIGURE 36. REPRESENTATION OF CONNECTIONS BETWEEN NODES.	78
FIGURE 37. BREAKING ONE SIDE OF THE DOUBLE LINKS BETWEEN NODES INSIDE THE CELLS.	79
FIGURE 38. CLASS DIAGRAM OF THE TEMPORARY CLASSES.	80
FIGURE 39. TEMPORARY FILE STRUCTURE FOR CONVERSION.	81
FIGURE 40. CONVERSION TOOL.	83
FIGURE 41. FINAL CLASS DIAGRAM OF THE CONVERSION TOOL.	84
FIGURE 42. EXAMPLE PROPERTIES DEFINITIONS FOR THE GEOSVIS LAYER.	86
FIGURE 43. GEOSVIS LAYER CLASS DIAGRAM.	87
FIGURE 44. OPENMAP GUI.	88
FIGURE 45. CHANGING DRAWING ATTRIBUTES OF A FEATURE.	89
FIGURE 46. THE GEOSVIS DRAWING ATTRIBUTES GUI.	90
FIGURE 47. TEST PATTERN ONE.	94

LIST OF TABLES

TABLE 1. A VPF FILE HEADER DESCRIPTION (DoD, 1996).	32
TABLE 2. MANDATORY COLUMNS FOR TOPOLOGICAL LEVELS (DoD, 1996).	38
TABLE 3. CRITICAL LINE SIZE FOR DIFFERENT SCALES.....	63
TABLE 4. CATEGORIZATION OF FEATURES FOR TESTING.....	93
TABLE 5. TEST RESULTS FOR PATTERN ONE.	95
TABLE 6. TEST RESULTS FOR PATTERN TWO.....	96
TABLE 7. TEST RESULTS FOR PATTERN THREE.....	97
TABLE 8. TEST RESULTS FOR PATTERN FOUR.	98
TABLE 9. TEST RESULTS FOR PATTERN FIVE.....	99
TABLE 10. CRITICAL FEATURES.	100

ABSTRACT

Geo-spatial databases have an overall performance problem because of their complexity and large size. For this reason, many researchers seek new ways to improve the overall performance of geo-spatial databases. Typically, these research efforts are focused on complex indexing structures and query processing methods to capture the relationships between the individual features of fully-functional geo-spatial databases.

Visualization applications, such as combat simulators and mission planning tools, suffer from the general performance problems associated with geo-spatial databases. This research focuses on building a high-performance geo-spatial database for visualization applications. The main approach is to simplify the complex data model and to index it with high-performance indexing structures. Complex features are reduced to simple primitives, then indexed using a combination of a disk-based array and B⁺-Trees.

Test results show that there is a significant performance improvement gained by the new data model and indexing schema for low to medium zoom levels. For high zoom levels, there is a performance drop due to the indexing schema's overhead.

PORTABLE HIGH-PERFORMANCE INDEXING FOR VECTOR PRODUCT FORMAT SPATIAL DATABASES

I. Introduction

Evolving satellite technologies increasingly help humankind to observe, discover, and learn much more about the earth than in past decades. Humankind is now using this data very heavily in daily life. Because the focus is the earth, the gathered data is very different and complex when compared to traditional data that database systems manage. In addition, there is a need to link traditional data, such as population of a city, to its geographical source on the earth. Here, as it can be easily seen there are two different types of data. The first, population of a city, is data that is commonly stored and is referred to as “traditional data” in this research. The second, geographical location of the city on the earth, is the data that is used to describe the location of the city on the earth, and is referred to as “spatial data.” Because of its differences from traditional data, Spatial Database Management Systems are used to manage spatial data instead of traditional database management systems. Of course like traditional database management systems, the main purpose of a spatial database management system is to deal with the effective and efficient management of spatial data (Shekhar, 1999). However, the complexity of spatial data itself makes management of spatial data very different and complex when compared to traditional databases as described by Ravada and Shekhar (Ravada, 2000; Shekhar, 1999). In addition to this complexity, the number of different users and applications are increasing daily, and performance becomes one of the vital issues for spatial databases. For this reason, research is being done to improve the performance of spatial database systems. This research is focused on improving the performance of spatial databases for applications that generally deal with the visualization of Earth-relative spatial data. Because Earth-relative spatial data is called geo-spatial data (Ravada, 2000), this research is focused on performance of geo-spatial databases in Geographic Information System (GIS).

1.1 Definition of Terms

Because this research deals with a special application area, the following key terms are defined:

Geographic Information System (GIS): “An organized collection of computer hardware, software, geographic data, and standard operating procedures for efficiently capturing, storing, maintaining, retrieving, analyzing, displaying, and reporting spatially referenced information” (DoD, 1996).

Spatial Database Management Systems: “Spatial database management systems aim at the effective and efficient management of data related to

- A space such as the physical world (geography, urban planning, astronomy);
- Parts of living organisms (anatomy of the human body);
- Engineering design (very large scale integrated circuits, the design of an automobile, or the molecular structure of a pharmaceutical drug); and
- Conceptual information space (a multidimensional decision support system, fluid flow, or an electromagnetic field)” (Shekhar, 1999).

Geo-spatial Database: A spatial database that stores and organizes Earth-relative (geographical) spatial data (Ravada, 2000).

Feature: A model of a real world geographic entity (DoD, 1996). Examples are rivers, roads, railroads, and boundaries.

Primitive: The smallest component of a spatial database, of which all features are composed (DoD, 1996). Examples are point (node), line (edge), and area classes to describe features.

1.2 Background

The importance of maps as a type of geo-spatial data has not changed for hundreds of years. In the past, they were generally used for navigation, and they were in printed form. However, evolving technologies in computer hardware and software have made maps possible to use in digital media format as digital maps for a wide variety of users. Today, not only government agencies but also companies, universities, and the public use digital maps. Although all users use the same digital map data, the applications are very different. For example, a biologist may be interested in the vegetation of a specific region while a transportation company may be interested in the road network of the area.

More importantly, digital maps are not just used to create or mimic the purpose of printed maps. They contain both geo-spatial and non-spatial data (traditional data). Geo-spatial data is complex and has a large volume in order to capture Earth-related data. That is why digital maps are organized as geo-spatial databases. This organization of geo-spatial data as digital maps allows different users to retrieve, alter, and query digital maps for different purposes.

Unlike traditional data, because gathering geo-spatial data requires very expensive special techniques such as space SAR (Synthetic Aperture Radar), there are few sources that produce GIS data. Generally, these sources are government agencies like the National Imagery and Mapping Agency (NIMA) and Environmental Systems Research Institute (ESRI), or commercial companies like Jeppesen and VISTA. Because they are independent from each other, their products have different formats. To solve this interoperability problem, the Open GIS Consortium (OGC) was established in 1994 (Cattenstart, 1999). The main goal of the OGC is to make standards for GIS data storage and sharing.

Although there are few sources that produce GIS data, the number of applications and users that use geo-spatial databases is increasing day by day with the help of evolving computer technology. These sources must support all applications with their GIS data. This obligation requires combining as much data as possible to support those applications. For example, the amount of data that NASA's Earth Observation System (EOS) sends to the Earth is approximately one terabyte each day (Chen, 1999). This illustrates that

the large volume of geo-spatial data is one of its properties and by itself is a good reason why a database management system must be used to organize GIS data.

What is the difference between geo-spatial and traditional databases? First, traditional data types such as integer, float, character, and string are not sufficient to specify the location of a geographical object, such as a dam on the earth. Using two traditional data types together may seem to solve this problem, but when complex objects and relations between these objects are considered, there is definitely a need for additional data types to describe geo-spatial data. Thus, geo-spatial databases include primitive types such as point, line, and area (Ravada, 2000; Shekhar, 1999).

Second, most of the features of a digital map require a combination of these primitives to become a meaningful object. For example, a road must be represented with a series of edges. This means that a geographic object may be composed of tens or even hundreds of primitives to become a feature. Also, even a very simple geographical object requires at least two, or most of the time three, dimensions to specify its location on the earth. Typically these dimensions are the latitude, longitude, and elevation data of the object.

Geo-spatial databases must also be examined to see their unique problems and differences when compared with traditional databases. First, geo-spatial databases require a complex data model to describe features and hide primitives from users. This data model must also keep all the relations between features and non-spatial data, and support all geo-spatial query commands.

Second, to reach a simple geo-spatial object such as a feature, which is composed of potentially hundreds of primitives, requires much more complex and different indexing techniques than are generally used in traditional databases.

Third, relations between geo-spatial objects are very different from those in traditional databases and occur in higher frequency, which affects query processing. Unlike traditional query commands such as “greater than” and “equal to,” geo-spatial databases must support more complex query commands like

“overlap,” “intersect,” “distance,” and “window.” To support these query commands, geo-spatial databases require more complex query languages, query processing, and cost estimation.

Fourth, geo-spatial databases must relate geo-spatial data with non-spatial data within this complex structure to support different kinds of applications. This is because most applications require application-specific traditional data in addition to geo-spatial data. For example, a biologist may want to keep information about different species, their population, and their habitat for a specific area. On the other hand, a meteorologist may want to keep statistical information like average rainfall per year for the same area.

In addition to these properties of geo-spatial databases, they must also support other functionalities of traditional databases such as integrity constraints and clustering. However, all of these affect the performance of geo-spatial databases. This is why most of the research in this area is focused on data models, indexing schemas, and query processing to improve the performance of geo-spatial databases. Because the overall GIS application area is very wide and GIS data is very large and complex, there are many research opportunities seeking to accomplish effective and efficient storage of the GIS data in geo-spatial database management systems.

1.3 Problem Statement

Geo-spatial databases, as discussed before, are organized to manage geo-spatial data for different applications. For this reason, their data model, indices, and query model are also organized for this purpose. When this generalization requirement is added to the previously discussed performance and complexity problems of geo-spatial databases, the overall performance of geo-spatial databases becomes the same for all applications. This is the result of a generalized complex data model, indexing schema, and query processing mechanism. The question “Do all the applications need all these complexities and generalizations?” may be asked. This depends on the query needs of an application. If it requires more query commands, it means that it requires more relations and complexity to describe those relations. Some

applications, however, just use geo-spatial data to visualize parts of a map to support their overall goal and do not require a complex, generalized query mechanism.

In addition, when general GIS applications are compared with applications that just visualize geo-spatial data, timing differences are found. For example, if a user wants to query a geo-spatial database for specific information, such as the highest mountain of a region, they can wait for the result because seconds are probably not important to them. However, if an application such as a flight simulation repeatedly updates the scene of a region, then the user probably cannot wait for a delay while querying a geo-spatial database to visualize the scene.

If generalization can be ignored for those applications that just visualize geo-spatial data, then complexity can potentially be decreased. This means that organizing a geo-spatial database just for these applications may improve their overall performance. As a result, instead of using image files or general geo-spatial databases, these applications can use visualization-oriented geo-spatial databases and obtain better performance. This research improves these kinds of applications by developing a supporting database infrastructure.

1.4 Objectives

The main objective of this research was to create a geo-spatial database infrastructure that performs better than general geo-spatial databases for visualization applications. To support this main objective, a set of visualization tools was implemented that read and visualize data for evaluation purposes. In addition to the main objective, an additional objective was to implement these tools in a platform-independent approach that allows users access on any platform.

The main objective required creating a new data model and indexing schema to store and retrieve using the new database. This objective also meant populating the new database with real data. As a result, a conversion tool was implemented to perform this task. It reads NIMA data and writes it to the new database according to the new data model and indexing schema. An implied requirement was to make the

geo-spatial database format as simple as possible because it serves as a key component of visualization applications.

1.5 Problem Scope

Because this research targets visualization applications, there are several restrictions. First, the new database is not a complex and fully functional GIS. It is designed to quickly respond to rectangle/window queries. It cannot be used to alter, edit, and query for other geo-spatial query information.

Second, there are two general approaches for a geo-spatial data model: vector and raster formats. This data model is based on the vector format and supports only that format.

Third, there are already many vector database formats and databases based on those formats. However, the conversion tool uses NIMA's VMAP database, which is built on NIMA's Vector Product Format (VPF) (DoD, 1996), to populate the new database. VPF represents geographic data in vector format by using nodes, edges, and faces in a file hierarchy. The VPF data model supports four level of topology for feature relations, and spatial and thematic indexes to reach data.

Fourth, instead of using a special database management system such as Relational Database Management System (RDBMS) or an Object-Relational Database Management System (ORDBMS) for the new geo-spatial database, binary random access files are used to store and index the new data model to support platform independence.

Fifth, the visualization tool is integrated into the open source application OpenMap. This decision was made because:

- OpenMap is implemented in Java™ and already supports platform independence,
- Many organizations use OpenMap for GIS applications,
- OpenMap provides a good interactive graphical structure to project map data,

- OpenMap already supports the VPF format, so it is usable for evaluation purposes,
- OpenMap supports a layered approach, and can be easily expanded to support other applications such as simulation and mission planning tools.

Finally, because the main goal is to improve performance, it is assumed that disk storage is available in quantity. Given today's low cost per gigabyte of storage, this appears to be a reasonable assumption.

1.6 Standards

The new geo-spatial database is based on the VPF database, defined by the Interface Standard for Vector Product Format, MIL-STD-2407, (DoD, 1996).

1.7 Research Focus

Because the goal is only to improve the performance of visualization applications, understanding how these applications use geo-spatial databases as compared to other geo-spatial applications is important.

Generally, these applications do not use a wide variety of spatial query commands. Instead, they query features for an intersection with a rectangular area, and visualize features that stay in that area. This property allows the construction of a more specific data model and indexing schema than general geo-spatial databases. The specialized database is optimized to respond only to this query command. Other general geo-spatial databases must capture all relations within primitives to respond to all geo-spatial query commands. This obligation makes their data model more complex and makes it very difficult to build fast indexing schemas. It is almost impossible to build an index on a feature that has one hundred edges, each of which is also composed of multidimensional data. For this reason, most geo-spatial databases use approximations of feature data such as Minimum Bounding Rectangles (MBR) instead of raw geo-spatial data.

Because of the complex data model and the approximation approach, all geo-spatial databases use filter and refine techniques based on feature approximations to retrieve data. However, this query process requires two steps to reach the correct data. In the first step, all features are filtered according to their approximations and in the second step geo-spatial data of features are examined.

The filter and refine technique returns entire features as a result of query processing. However, this is not a correct response for many visualization applications. For example, while querying for boundary features and the interest area is the city of Dayton, the query process may also return the feature that represents Ohio state boundaries or even larger features like the entire political boundaries of the U.S.

These applications, which just visualize GIS data, generally read GIS data from the geo-spatial database. They do not perform any deletion from, or addition to, GIS data. This property allows the application of more specific indexing and clustering methods.

These observations help to create more optimized database in which, the following simplifying assumptions we made:

- The database does not need a complicated data model and all the relations between features. Features can be broken down into small primitives that make them easy to index because one hundred continuous edges and one hundred separate lines are the same for visualization. This allows using row geo-spatial data for indexing instead of feature approximations. A two dimensional indexing schema based on latitude and longitude values of primitives can be used to reach the primitives in the interest area faster.
- When feature approximations are not used for indexing, the filter and refine technique is no longer needed. All data is retrieved in one process, without the filter and refine technique.
- Query processing need only return small, related primitives instead of large features.
- More specific and stable indexing and clustering techniques can be used.

In this approach large features are broken down into node and edge primitives. A combination of array and B+Tree data structures are then used to index the multidimensional geo-spatial data (latitude and longitude coordinate values). Because there is no precision limit on coordinate values, the data space is divided into small cells and the B+Tree data structures are used to index these small cells. An array structure is used to point to multiple B+Tree data structures. As a result, a kind of sparse matrix structure is used to reach a primitive by its coordinate values. With this indexing structure, it is very easy to find a unique cell by its latitude and longitude values in the data space. The array data structure allows reaching related B+Trees according to a primitive's longitude coordinate, and the B+Tree is then used to reach the related cell according to a primitive's latitude coordinate. As a result, the overall structure becomes a matrix structure without empty cells. A more complete description of this structure is given in Chapter 3.

1.8 Summary

The complexity and generalization problems of the geo-spatial data affect the performance of all geo-spatial databases. As a database management system, geo-spatial databases provide many geo-spatial query commands to their users to query geo-spatial data. However, storing complex geo-spatial features makes this process difficult. For this reason, significant research is focused on improving the performance of geo-spatial databases with this kind of complexity. On the other hand, there are some applications that do not need this level of complexity because their main purpose is to visualize the geo-spatial data. Because of the generalization problem, their performance is also affected by the complexity of the geo-spatial databases. Many of these applications need performance more than other applications because they use visualization to support their main goal. As a result, any delay at reaching geo-spatial data degrades their usefulness.

If the generalization is ignored for these applications then complexity can be significantly decreased, which helps to create visualization specific geo-spatial databases. This research focuses on the creation of a new geo-spatial data model and indexing schema to improve the performance of these applications.

II. Literature Review

2.1 Introduction

In the information era, evolving technologies increase both the need to access Earth related geographic information and the variety of applications that use that geographic information. In the past, government agencies or large companies have commonly used this geographic information, generally for navigation, and information was in the form of printed media such as maps. Nowadays almost every agency, company, or individual uses geographic information for very different purposes, and information is not in the form of printed media; it is mainly electronically stored and managed by databases. In addition, many different users generally share geographic information for different applications. A meteorologist may examine a thunderstorm that is approaching a town, a veterinarian may examine an animal disease according to environmental conditions, or a biologist may search for living organisms in a river. This general application area used today is called Geographic Information System (GIS), and GIS data is stored in Geo-spatial Database systems. The objective of this literature review is to examine the general accomplishments and the problems of spatial database systems.

Like traditional database management systems, spatial database management systems must also deal with effective and efficient management of GIS data (Shekhar, 1999). Because of the complexity and the large universe of the spatial data, spatial database management systems have the same problems as traditional database management systems. These problems, such as complex data models, indexing schemas, and query processing of spatial data, generally affect the performance of spatial databases. Therefore, much research is being done to improve the performance of spatial database systems. Because the aim of this research is to improve the performance of retrieving spatial data from spatial database systems for visualization purposes, general information and performance problems of spatial database systems are discussed in this literature review.

The review is organized into four sections. The first describes and discusses the general accomplishments, problems and efficiency in spatial databases. The second examines an example spatial

database system, the Vector Product Format (VPF). The third deals with visualization aspects of spatial database systems, and the fourth summarizes the review.

2.2 Spatial Databases

The aim of spatial database management systems is to store, retrieve, and manipulate spatial data (Orlowska, 2000). Their purpose can be summarized as making the spatial data management easier and more natural for the users of applications that are dealing with spatial data (Ravada, 2000). However, spatial data itself is very different from the data that is stored in traditional database management systems. Spatial database management systems are managing the data that is related to (Shekhar, 1999):

- A space such as the physical world (geography, urban planning, astronomy);
- A part of living organisms (anatomy of the human body);
- Engineering design (very large scale integrated circuits, the design of an automobile, or the molecular structure of a pharmaceutical drug); and
- Conceptual information space (a multidimensional decision support system, fluid flow, or an electro-magnetic field).

The main application areas of spatial database management systems can be listed as Geographic Information Systems (GIS), Computer Aided Design (CAD), Computer Aided Manufacturing (CAM), Multi-media Information System (MMIS), and Data Warehousing (DWH) (Shekhar, 1999; Ravada, 2000).

2.2.1 Geographic Information Systems (GIS)

This research is focused on only one application area: GIS. GIS is generally used to manage Earth-relative spatial data (Ravada, 2000). It can be defined more precisely as a “computer based information system that enables capturing, modeling, manipulation, retrieval, analysis, and presentation of geographically referenced data” (Fonseca, 1999; Worboys, 1995). Geo-spatial database systems are used

to manage GIS-data. Hereafter in this research, the term “spatial database management systems” will mean “geo-spatial database management systems.”

The Earth Observing System (EOS) is one of the best examples of the GIS application area. It is sponsored by NASA with the purpose of monitoring the Earth for the study of global environmental change. More than 13 satellites will continuously and systematically observe Earth for a minimum of 15 years. A large volume of heterogeneous data will be observed, such as land, vegetation, hydrology, atmosphere, oceanography, cryosphere, and solar radiation. Managing this variety of data is the aim of spatial database management systems (Yin, 2000).

From the EOS example, it can be seen that GIS is a very wide application area and is composed of a very large volume of heterogonous data. GIS-data is generally collected by different custodians and composed of separate data collections (Orlowska, 2000). The numbers of custodians are very limited when compared to the number of users because of the high technological and financial needs required to collect GIS data. However, most applications require combining different datasets for different purposes. This integration process may be very helpful, but at the same time it is very difficult.

2.2.2 The Open GIS Consortium (OGC)

The Open GIS Consortium was established in 1994 to overcome the difficult interoperability problem (Cattenstart, 1999). OGC and its purpose may be described, as “an open, industry-wide consortium of GIS vendors and the users who are attempting to facilitate interoperability by proposing standards for GIS knowledge interchange” (Cranston, 1999). The consortium has proposed, “OGC Simple Features Specification For SQL” (Cattenstart, 1999), and “Web Mapping Specification” (Yin, 2000) as specifications that intend to standardize the use of spatial databases.

2.2.3 Properties of Spatial Database Management Systems

Spatial databases are designed to manage two types of data: spatial data (spatial component) and non-spatial (thematic component) or attribute data (Cicerone, 2000). The spatial data is the data that is

related with the geography or Earth location such as longitude, latitude, and height. Non-spatial data is related to the properties of spatial data or geographic primitives. For example, the coordinate values of a bridge are the spatial data because they describe the location of the bridge on Earth. However, other information about the bridge like its architectural style and manufacturer information are the non-spatial data. All spatial query operations work on spatial data and need to separate a feature according to its spatial data. When complex features that can be composed of hundreds of primitives are considered, this obligation requires complex data types, structures, and relations to identify features. This is why spatial data is complex when compared with non-spatial data.

Because of this complexity of spatial data, it is generally difficult to combine spatial and non-spatial data types in one database management system. Most often spatial data is stored in files managed by a file management system and non-spatial data is stored in a traditional database management system, generally a Relational Database Management System (RDBMS) (Ravada, 2000). Spatial data files are generally organized in a directory and file hierarchy to increase performance. For this reason, directory and file names have strict restrictions, and no duplicated file names are allowed. Spatial indexes can be built upon this directory hierarchy to boost performance. This hierarchy also serves as a kind of clustering for spatial data. Objects related according to geometry space are stored in the same directory and objects near each other are stored in the same file (Shekhar, 1999). The Vector Product Format (VPF) file hierarchy is a good example of this approach. For example, all transportation data such as railroads and roads are stored in a subdirectory, which is called “trans.”

However, this approach has several drawbacks in the areas of database integrity and complexity for the user. The ideal solution is combining the two data sets in to one database management system. Here, Object Relational Database Management Systems (ORDBMS) and Object Oriented Database Management Systems (OODBMS) are much more suitable than an RDBMS because they allow the specification of user defined attributes to describe spatial data. This approach has several benefits. First it helps to store and manage the data with an enterprise wide DBMS. Second, it reduces the complexity of the system. Third, it increases the interoperability of the different GIS systems (Ravada, 2000).

When the second approach is chosen, spatial databases must support some functionality for the spatial applications as a database management system. These requirements of spatial databases are addressed by Ravada (Ravada, 2000) as:

- Classification of space,
- Data model,
- Query language,
- Query processing,
- Data organization and indexing.

These requirements as applied to this research will be discussed in more detail in later chapters.

2.2.4 Traditional Databases Versus Spatial Databases

As has been discussed, spatial database systems differ from traditional database systems in the spatial data itself. The general data types of a traditional database system are primitives like integer, float, character, and date. In addition, the operations on these data types are simple arithmetic or logic operations. However, spatial data types are more complex than these primitive data types. They are generally composed of points, lines, faces, or areas to indicate any Earth-relation data. This complexity simply required catching the relations between primitives because operations on spatial data are even more complex than the operations on traditional databases (Ravada, 2000). Because the Earth is large and has many detailed features, the other important difference is the large volume of spatial data as noted in the EOS example.

The main differences of spatial databases that separate them from traditional databases are described by Egenhofer (1993) as:

- Geometry: It is the main and the most important part of spatial databases. It is also defined as “space taxonomy” by Shekhar (1999). Geometry mathematically defines an

object. Examples of the geometries are set-based (elements, element quantity), topological (points, lines, surfaces), Euclidean (coordinate system), metric (distance, measure) and network (nodes, connectivity).

- **Distribution of Objects in Space:** When we think spatial objects like cities or lakes, they are always irregularly distributed. In addition, their sizes vary large in extent, like a big city and a small city.
- **Temporal Changes:** Because spatial objects have also non-spatial attributes associated with them, they generally have a temporally changing behavior, like a dam's water capacity.
- **Data Volume:** GIS applications generally deal with the large spatial data sets.

The difference between spatial databases and traditional databases may be summarized in three main areas. First, any spatial database must support spatial data types like point, line, face or area. Second, it must support the query operations like intersect, covers, and distance on these spatial data types. Finally, it must support indexing schemas to increase the performance of the database to reach spatial data (Ravada, 2000).

2.2.5 Data Model

The main purpose of the spatial data model is to hide the details of the data storage. For this reason, the best approach is to use abstract data types (ADT) in the modeling process. The user defines the data model and operations that can work on it by using ADTs. ORDBMSs and OODBMSs allow the user to define ADTs, so they are the most common DBMSs used in spatial applications.

There are two common data models that are generally used in spatial databases, one is the field-based model and the other is the object-based model (Shekhar, 1999). The field-based data model, generally called as raster data model or field-based raster data model, “views the geographical reality as a set of spatial distributions over the geographical space” (Camara, 1999) and it is generally used to store

image data such as topography, vegetation maps and satellite images. The common operations on field-based models are local, focal and zonal. The object-based data model “represents the world as a surface occupied by discrete, identifiable entities, with a geographical location (with possible multiple geometric representations) and descriptive attributes” (Camara, 1999) and is generally used to store human-built features like roads and buildings. Examples of common operations on object-based data models are distance and boundary. The object-based data model generally consists of data types, such as points, lines, and polygons.

OGC released “Simple Features Specification For SQL” (Cattenstart, 1999) to make the spatial data model (Figure 1) more standard. This data model has point, line, curve, and surface primitives, and it can be seen that these primitives are not used just to hold spatial data values but they are also used to inherit more complex primitives and to catch relations between these primitives. This is why spatial data primitives such as points and lines are required for spatial databases instead of traditional data types such as float and integer.

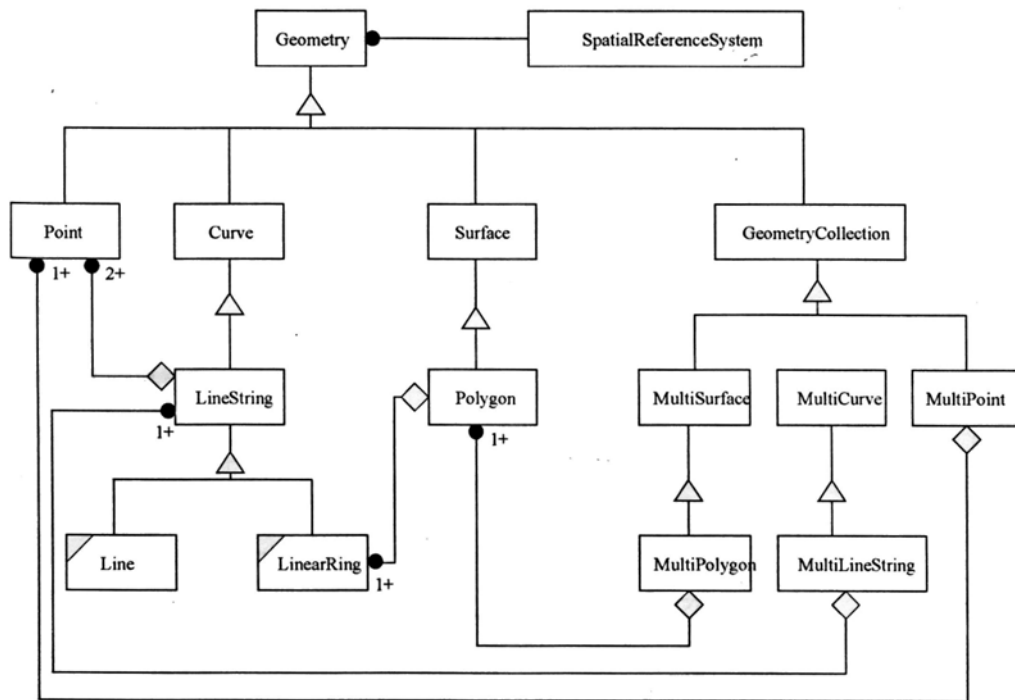


Figure 1. The data model of the OGC specification (Cattenstart, 1999).

As discussed in detail by Cattenstart (Cattenstart, 1999), the data model in Figure 1 has some problems defining some complex features and their relations. There are some inconsistencies between this specification and major commercial packages such as Environmental Systems Research Institute Spatial Data Engine and the ORACLE Corporation Spatial Data Option. Even with the OGC specification there are still standardization problems as noted by Camara (Camara, 1999): “The OpenGIS standard has not been defined in a formal and unequivocal way, and therefore, there are definition problems and competing alternatives for mapping existing GIS systems into proposed standard.”

As noted by Fonseca (Fonseca, 1999), integration is still an important problem despite the OGC Simple Features Specification: “Now interoperability is turning into an integration science.” There are many research efforts to integrate different spatial database systems as mentioned by Orlowska (Orlowska, 2000), Fonseca (Fonseca, 1999), Camara (Camara, 1999), and Cattenstart (Cattenstart, 1999).

2.2.6 Query Processing

The spatial query commands and complex primitives, such as faces and areas, make query processing very costly when compared to traditional databases. Retrieving an entire large primitive object into memory is time consuming. Spatial query computations are generally complex and costly; the more complex the primitive object is, the more processing time and power required to compute the query (Ravada, 2000). Generally, an approximation method is used to overcome these problems. Instead of indexing the entire feature primitive by primitive, an approximation of the feature is indexed. Common approximation geometries for spatial data are minimum orthogonal bounding rectangle/box (MOBR) and multiple bounding rectangles/boxes (Zhu, 2000; Shekhar, 1999; Ravada, 2000).

The main purposes of the approximations are to increase query performance and to save memory and computing power. For example, in Figure 2 MOBRs of the primitives D and B do not intersect, so the primitives cannot intersect. This means that an application can eliminate primitive D easily for an intersection query without loading the entire primitive to the memory. The main idea is “if the approximations of objects A and B do not satisfy a relationship, then that relationship cannot be satisfied

between the objects A and B” Ravada, 2000. However, MOBRs of the primitives A and B intersect, but the original primitives do not intersect. This requires extra processing to omit feature A even though it is not needed.

In addition, MOBRs may not represent complex polygons adequately. An improved version of MOBR approximation is the n -corner convex polygon approximation, and its purpose is the adequate approximation of the complex polygons (Zhu, 2000). Figure 3 shows an example of n -corner (5-corner) approximation. However, increasing n requires extra computing and storage space, so this approach may not be suitable for large numbers of n .

Because the cost of spatial queries that are based on complex polygons is very high, approximations are used to increase performance. For this reason, spatial queries generally use a two-step filter and refine technique (Theodoridis, 1998; Park, 1999). In the filter phase, approximations such as MOBR are used to filter the irrelevant objects. In the refine phase, exact primitive geometry is used to refine the remaining primitive objects for the query (Zhu, 2000; Park, 1999; Shekhar, 1999).

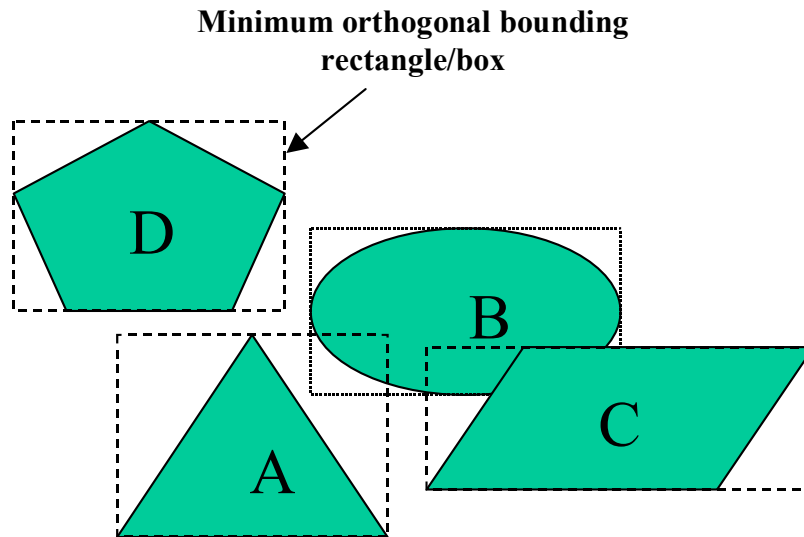


Figure 2. Approximations by Minimum Bounding Rectangle/Box.

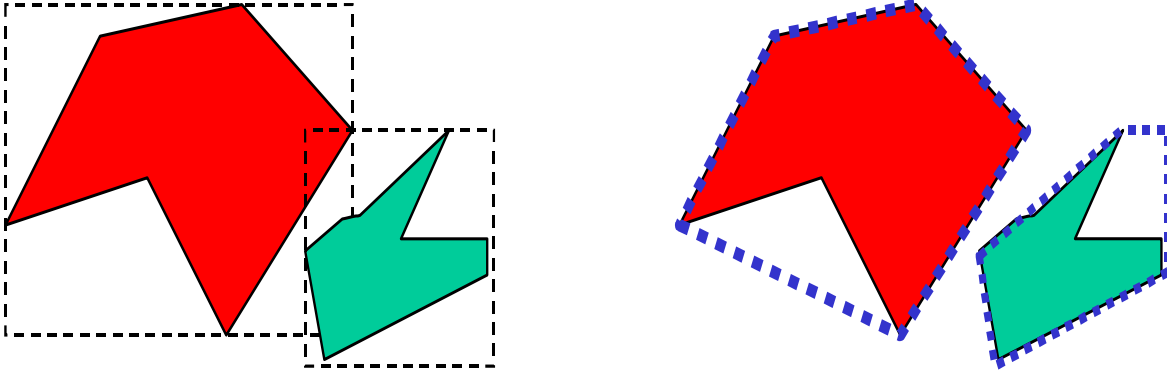


Figure 3. *N-corner convex polygon approximation.*

The idea of using filter step with approximations is to eliminate as many false spatial objects as possible before performing more costly operations on the actual representations of the complex spatial objects (Bertino, 1999). Multidimensional indexes based on the approximations of the objects are used in the filter step and geometric techniques for spatial operators are used in the refine step (Theodoridis, 1998). Because all the filtered primitives must be checked, the second step is the most costly and time-consuming procedure. The steps of this technique are demonstrated in Figure 4; the query searches the primitives that intersect with primitive B. MOBR approximation helps query processing to eliminate primitive D without retrieving the entire primitive into memory.

Like relational databases there are two types of query categories in spatial databases: selection queries (window, range-queries) and join queries (spatial-join queries) (Theodoridis, 1998; Ravada, 2000). Selection queries query spatial primitives according to a known spatial primitive that requires scanning all spatial primitives using indexes. Join queries generally search the two tables for an object pair matching by using nested loop and tree matching (Shekhar, 1999).

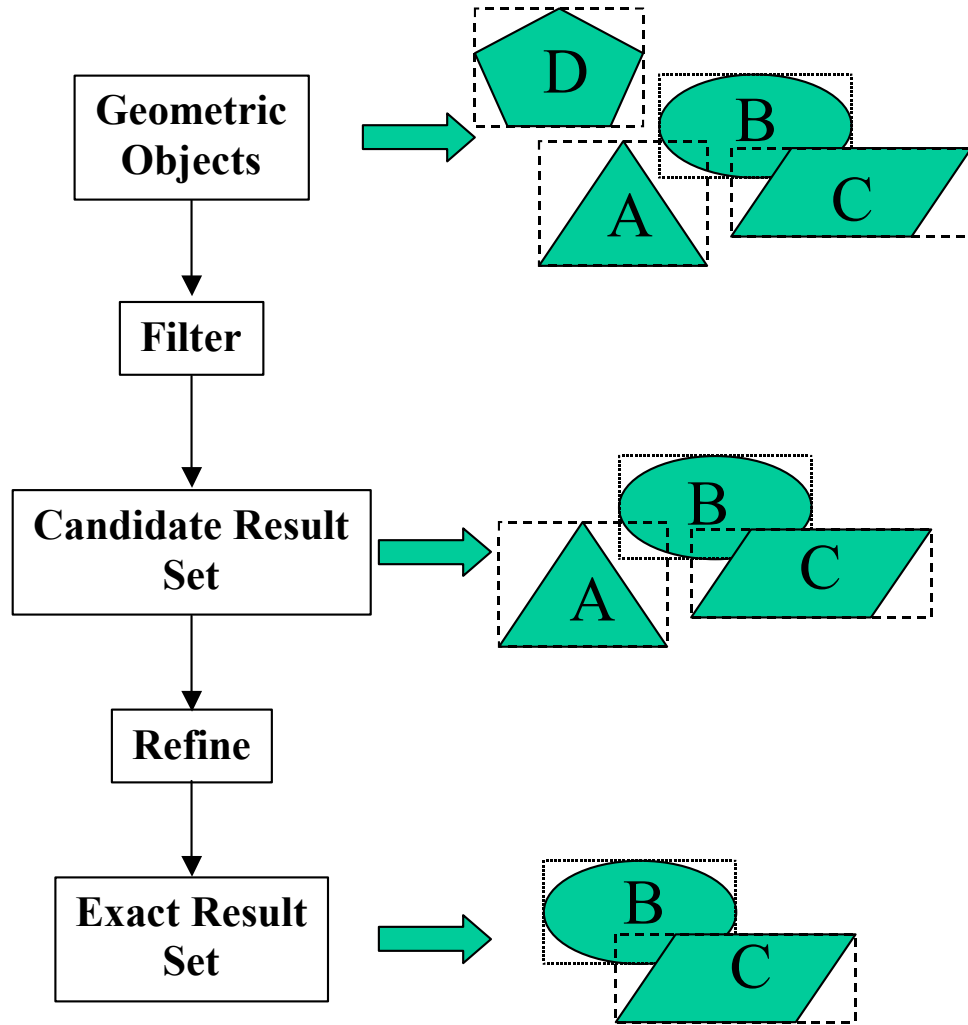


Figure 4. Using approximations at filter and refine techniques.

“Find all the hospitals within 20 miles of the Dayton International Airport” is an example of a selection query, and “Find all the airports that have one or more hospitals within 20 miles” is an example of a join query. In the first query, all the hospitals will be searched according to Dayton International Airport’s location. Here Dayton International Airport is the window object, and query retrieves all the primitives that satisfy the spatial operation based on the window object. Range queries are generally implemented by a downwards traversal of an R-tree index (Ravada, 2000). For the second query, all the hospitals will be searched according to the airports’ location, and this query requires processing the tables of the hospitals and the airports at the same time. Simply stated, join queries combine two different data

sets and retrieve the information from their Cartesian product. Because of the large and complex geometric primitives and multidimensional indexing, processing a join query by combining two separate data sets is very costly, but it is also very fundamental for spatial databases. Join queries can be implemented by applying a synchronized traversal on both R-tree indexes (Ravada, 2000).

2.2.6.1 Query Language

Spatial databases must support a query language to access and manipulate spatial data. This query language must be intuitive and easy to use, and at the same it must support spatial data types and spatial query commands (Ravada, 2000). OGC released the “Simple Features Specification For SQL” to support a common query language for spatial databases (OGC, 1998). In addition, there is a mandatory query language as OGC Common query for the OGC Catalog Interface Specification (Rao, 2000). Both languages are based on SQL and support nested Boolean queries, text matching operations, temporal data types, and relational operations (Rao, 2000). An example SQL query with spatial operators and corresponding query tree is as shown in the Figure 5 (Shekhar, 1999).

2.2.6.2 Other Accomplishments and Research on Query Processing

In the field of spatial databases, most of the current work and research is focused on data models, indexing, and query processing steps to improve performance. There is some research on spatial query optimization and cost estimation. Park discusses three different query optimization strategies (Park, 1999) and Brinkhoff and Theodoridis discuss different strategies for the cost estimation of spatial query operations (Brinkhoff, 1993; Theodoridis, 1998; Theodoridis, 2000).

```

SELECT  L.name, Fa.name
FROM    Lake L, Facilities Fa
WHERE   Area(L.Geometry) > 5 AND
        Fa.type = campground AND
        Distance(Fa.Geometry, L.Geometry) < 20

```

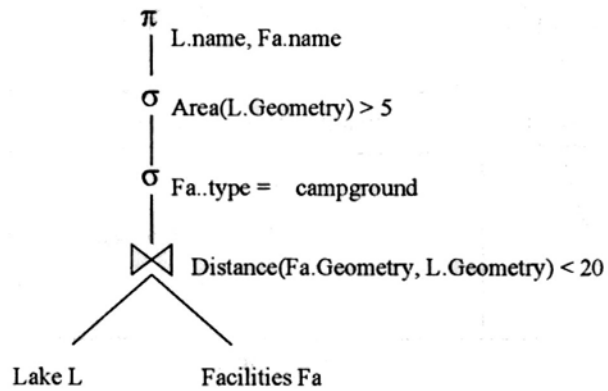


Figure 5. A simple query tree for a spatial database (Shekhar, 1999).

2.2.7 Indexing

The purpose of spatial indexing is to increase the performance of spatial selection retrieval to facilitate spatial query processing. Like spatial query processing, spatial indexing also uses approximations (Bertino, 1999). MOBRs and grid approximations are common approximation types to represent a spatial primitive. By means of the MOBR approximation, each primitive is represented by the smallest orthogonal rectangle enclosing its boundaries. Grid approximation divides the space into small cells and the cells that intersect the primitive represent the primitive. Approximations are very simple graphical objects like a rectangle when compared to primitive object itself to help the filter process in spatial query processing (Ravada, 2000).

Like traditional databases, a spatial access structure partitions the data space and associates the data with subspaces. However, because of the complexity of the spatial data model and its associated representation problems, it is very critical for spatial databases to properly partition data (Bertino, 1999).

There are three main indexing structures used by spatial databases to associate spatial objects with the data space. The following information is summarized from Bertino (Bertino, 1999):

- The transformation approach:
 - Parameter space indexing: Spatial objects that have n vertices in a k -dimensional space are mapped into an nk -dimensional space. In this structure spatial objects are represented as points in nk -dimension, so it becomes easy to store the object in a point index. For example, a rectangle defined by two points in a two-dimensional space is represented as a point in four-dimensional space. This structure can be stored in a multidimensional indexing structure. However, spatial relations between the objects may be broken.
 - Mapping to a single attribute space: The space is divided into equally sized grid cells, and then grid cells are mapped into one-dimensional space by using curve-filling methods such as Z-order, row, and Peano-Hilbert. Figure 6 shows how the space-filling curves are used to linearize a multidimensional space (Shekhar, 1999). After mapping from multidimensional space to one-dimensional space, B^+ -trees can be used to index the cells.
- The non-overlapping native space indexing approach:
 - Object duplication: Like grid cells, a k -dimensional data space is partitioned into disjoint subspaces. A spatial object is represented by the subspaces that intersect the object. An object identifier is duplicated in every subspace.

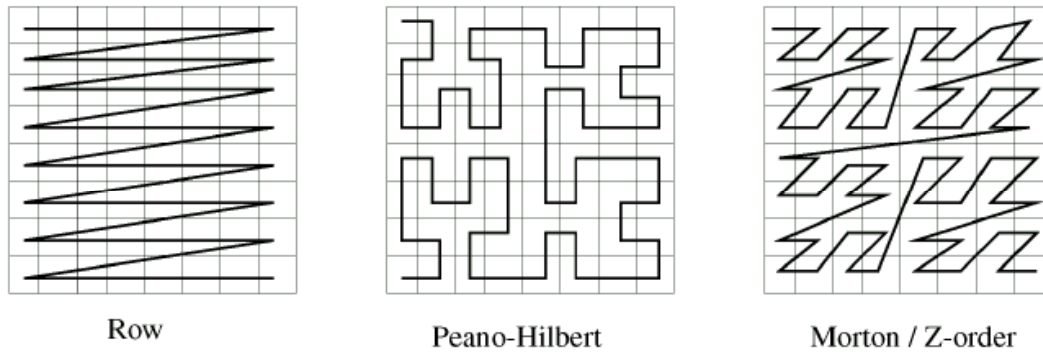


Figure 6. Curve filling methods (Shekhar, 1999).

- Object clipping: This approach is a variation of object duplication. Instead of duplicating the object identifier for every subspace, the object is decomposed into subobjects. In these approaches, objects or the object identifiers are duplicated, so extra storage is required.
- The overlapping native space indexing approach: This approach is very similar to the non-overlapping indexing approach. The main idea is to partition the entire data space into small manageable subspaces, but object overlapping is allowed for a proper representation. Hierarchical indexes help to store spatial objects in subspaces in their native space.

Spatial indexing methods can be also categorized as indexing methods for points (PAM, point access methods) and indexing methods for non-point objects (SAM, spatial access methods) (Theodoridis, 2000). BANG files and LSD-trees are the examples of PAMs, R-trees and Quadrees are the examples of SAMs.

There is another categorization in addition to the above-mentioned categories. This categorization is based on indexing schemas as one-dimensional and multidimensional. In general, after mapping from multidimensional space to one-dimensional space by using space-filling curve methods, generally B^+ -trees can be used to index one-dimensional spatial objects. However, because of the multidimensional structure of spatial data, many indexing methods that support multidimensional Euclidean space have been explored

(Shekhar, 1999). One of the first multidimensional indexing methods for extended objects is the R-tree (Bertino, 1999; Theodoridis, 1998; Shekhar, 1999). It is still one of the most popular and the efficient indexing methods. However, there are many variants of R-trees such as R^* -trees and R^+ -trees that try to improve the performance of the R-tree indexing method.

The R-tree

Guttman extends the R-tree method from the B^+ -tree to support the d dimensions, which is required for spatial databases (Shekhar, 1999). Like B-tree, R-tree is also height-balanced and composed of intermediate and leaf nodes. Minimum bounding rectangles (MBR) represent spatial objects in an R-tree structure. A leaf node in the R-tree is in the form of (oid, MBR) , where oid represents the spatial object and MBR represent the minimum bounding rectangle of the spatial object. An intermediate node is in the form of (ptr, MBR) , where ptr points to a lower level node, and MBR is the minimum bounding rectangle that encloses all lower level nodes Theodoridis, 2000.

General properties of the R-tree are also specified by Theodoridis (Theodoridis, 2000), where M equals the maximum number of entries in a node and m equals to the minimum number of the entries in a node and $m \leq M/2$:

- Every leaf node contains between m and M entries, if it is not the root.
- Every intermediate node has between m and M children, if it is not the root.
- All leaves appear in the same level.

The hierarchical structure of the R-tree allows partitioning data space recursively into small subspaces, and the smallest unpartitioned space is generally equal to the physical page space for effective storage. An R-tree is a dynamic hierarchical spatial index, which is sensitive to the order of data insertion. Different insertion sequences of the same data set may cause the tree to behave differently. Also node splitting and merging may be involved when spatial objects are inserted and deleted (Bertino, 1999). Figure 7 shows how an R-tree maps the data space.

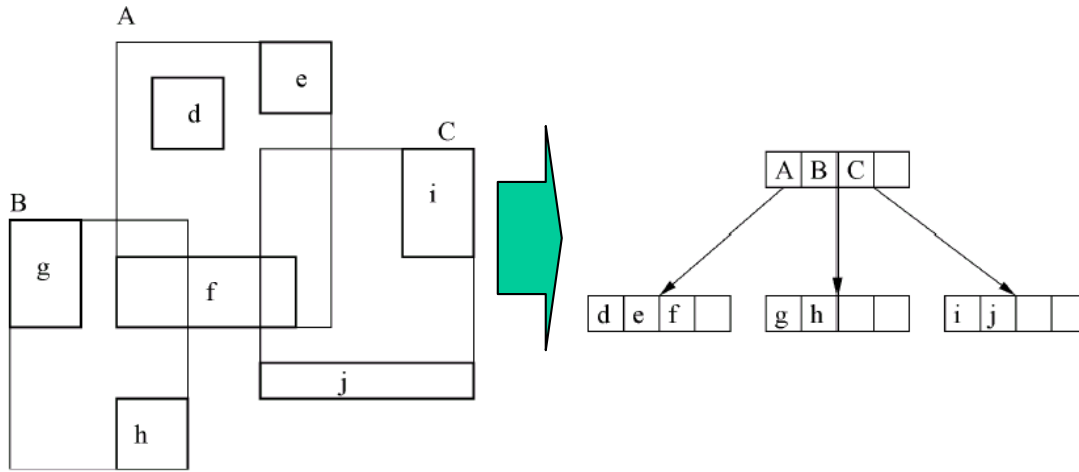


Figure 7. Mapping features using an R-tree indexing structure (Bertino, 1999).

There are many variants of the R-tree which try to improve its performance. R-link is used for concurrency control of spatial databases during insert, delete, and search operations (Shekhar, 1999). R-link has additional sibling pointers to the structure of the R-tree to check the modifications. For static databases where the subspaces are well known and seldom change, packed R-trees are generally used (Mutenda, 2000). An R^+ -tree decreases the redundancy of nodes and guarantees disjointed nodes. The R^* -tree has a more complex structure than the R-tree and improves the grouping algorithm to build efficient tree structures dynamically based on the MBRs of the features (Theodoridis, 2000). Parallel R-trees are useful in single processor/multiple disk systems. Because disk access is more important than CPU power, parallel R-trees organize nodes on different disks to distribute the load across the all disks (Mutenda, 2000). In this structure, inner nodes of the tree structure are stored at the master disk, and the leaves that point to the features are distributed to the slave disks. The query is first processed at the master disk on inner nodes and then the features are read from the slave disks in parallel.

Research on index structures is focused on high-dimensional index structures. As discussed previously, there are many varieties of multidimensional index structures. These structures are generally organized for low-dimensional space. In high-dimensional space, these structures are affected by different variables and their performance decreases significantly (Berchtold, 2000). High-dimensional index structures aim to improve the performance of the indexes for high dimensions where multidimensional

indexes have side effects and performance problems. TV-tree (telescope vectors), SS-tree, SR-tree, and the X-tree are the examples of high-dimensional index structures, which are based on R-tree indexing (Berchtold, 2000).

2.2.8 Commercial Systems

After OGC released the Simple Features Specification for SQL OGC, 1998, some commercial systems appeared on the market based on the specification. These are Environmental Systems Research Institute (ESRI) Spatial Data Engine 3.0.2 for ORACLE version 7, ORACLE Corporation Spatial Data Option (SDO) 7.0 for ORACLE version 7.3.3, Spatial Data Cartridge (SDC) 8.0 for ORACLE 8.0 (Cattenstart, 1999), Informix 2D Datablade, and IBM's DB2 Spatial Extender (Ravada, 2000). These object-relational database systems allow applications to integrate spatial data with non-spatial data in the same database system. These ORDBMS also provide user-defined indexes and query language support, so that spatial data can be managed like non-spatial data. Most of these database systems also support query optimizations for spatial operations (Ravada, 2000). The OGC data model and its comparison with the other commercial data models were evaluated by Cattenstart (Cattenstart, 1999) who suggests that commercial data models are not completely OGC feature compliant. Some deviations come from the application developer concept and the way geometric entities are handled in their systems.

2.3 Vector Product Format

The Vector Product Format (VPF) is described as "A standard format, structure, and organization for large geographic databases that are based on geo-relational data model" (DoD, 1996). Any application can use the VPF format to access vector format digital geographic data without conversion. VPF represents geographic data in vector format by using nodes, edges, and faces. VPF also supports spatial and thematic indexes to improve the performance of the database. VPF supports a geo-relational data model with combinatorial topology and set theory. A VPF-compliant database product must include and interpret all mandatory VPF tables and columns. All the information that is discussed in this section is based on the military standard MIL-STD-2407 (DoD, 1996) unless otherwise noted.

2.3.1 VPF Characteristics

The aim of the VPF model is to support digital geographic databases with a flexibility to encode spatial data, and its characteristics can be summarized as:

- Sheetless database support: VPF supports accessing, querying, and retrieving of the data that extends across tile boundaries. Tiles are used to partition the coverages for spatial decomposition.
- Neutral format: VPF has a product-neutral format that an application must use. This format supports several levels of topologies.
- Attribute support: Attributes are stored in tables that can support both simple and complex features.
- Data dictionary: VPF allows the storage of data definitions for information stored in the database.
- Text and metadata support: VPF has multi-language text support both for features and feature attributes.
- Index file support: VPF supports both spatial and thematic indexes to improve performance. While spatial index files can be used to reach a primitive using its coordinate values, thematic index files can be used to reach the primitive using its row IDs.
- Direct access: A VPF-compliant application can read geographic data without any external database support or conversion.
- Flexible, general-purpose schema: VPF provides some flexibility in how the digital geographic data is encoded. It supports coordinate pairs, triplets and multiple scaling.
- Data quality: VPF provides data quality reporting and representation.

- Feature definitions: VPF creates logically consistent and complete products by organizing features and thematic attributes.

2.3.2 VPF Data Model

The VPF data model can be broken down into three areas: data organization, data model components, and data quality. The data organization simply organizes physical structure of the VPF data. The VPF data model components capture the real world geographic entities and encode them in the VPF data structure. Data quality information is provided by the VPF format for the maintenance of VPF data.

2.3.2.1 Data Organization

Because VPF databases do not require enterprise DBMSs, the physical structure of VPF data is organized hierarchically as directories, tables, and indexes. In VPF databases all file and directory names must be in the lower case.

2.3.2.1.1 Directory

Directories organize all the VPF tables in hierarchical order. There are some restrictions on how to use and name the tables.

- Each file must be contained in exactly one directory.
- Files names must be unique within a directory. However, they need not be unique between directories.
- Pathnames can be used to reference a file in the following form:

`<directory name><separator><file name>`
- VPF format uses backslash character “\” as a directory separator.
- Pathnames can be combined since a directory may contain other directories.

2.3.2.1.2 Tables

In the VPF format, tables model all geographic entities. VPF has a mandatory common structure for all tables and organizes data content in these structures. A VPF table structure must stick to at least the basic structure and may have references to additional structures like narrative tables and value description tables. In addition, a VPF table may reference variable length index and spatial index tables.

A VPF table is composed of three parts: a table header, a row identifier, and the table contents. The header contains metadata about the table and column definitions. Data contents are organized into rows and columns as in a relational database. Every column must have a unique name that defines/separates the column from the other columns. In addition, each row has a unique row identifier that starts from 1 and increases sequentially without any gap. An example of the VPF table structure is given at Table 1.

There is no specific column ordering in VPF tables. There are mandatory and optional tables and columns in VPF, and any VPF-compliant product must include them in its database.

The information about the real world objects, entities, or features, is stored in attribute tables that are organized as tables in the relational database systems. A VPF table or column may have an associated narrative table that simply describes the table/column.

2.3.2.1.3 Indexes

A VPF table may reference spatial, thematic, and variable-length indexes. Spatial indexes point to row data according to the value of coordinate columns. Thematic indexes point to a row data according to the value of non-coordinate columns. If a table has a variable-length coordinate string column or a variable-length text string column, a variable-length index table associated with that column must be present.

Table 1. A VPF file header description (DoD, 1996).

Table Header	
Metadata and column definitions:	
a. Table description	
b. Narrative table name (optional)	
c. Column definitions:	
Column name	
Field type	
Field length	
Key type	
Column textual description	
Optional value description table name	
Optional thematic index name	
Optional column narrative table name	
ID	Table Contents
Indicates the starting position of each row.	The data composing the table that match the column definitions.

There are four categories of spatial decomposition in a VPF database: the tile directory, spatial indexes, the MBRs of the primitives, and coordinate values of the primitives. Spatial indexes are based on MBRs of primitives, and they are not mandatory. The inner structure of a spatial index file is based on an adaptive grid binary tree. This structure is a grid-based binary tree that holds the decomposed primitives. This structure can handle any type of spatial primitive such as a point, line, or area. This structure is very similar to the R-Tree structure previously discussed.

2.3.2.2 Data Model Components

The VPF data model is structured in four levels as shown in Figure 8. The real world geographic entities are captured as feature classes, which are defined by the VPF primitive and attribute tables. Feature classes form the lowest level in the hierarchy and the combination of feature classes form coverages. Coverages make up libraries and libraries make up the database. In general, coverages define the relationships between features, and databases and libraries support access to them.

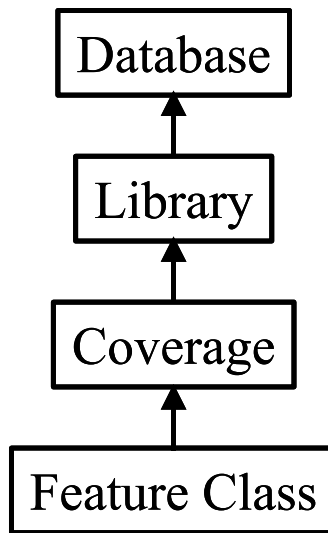


Figure 8. VPF data model levels (DoD, 1996).

2.3.2.2.1 Feature Classes

A feature is a combination of one or more simple primitives or complex primitives. Railroads and roads are examples of feature classes. Features are grouped into feature classes and stored in feature tables. Every feature has a unique name such as “roadl” and “contourl,” which defines the feature. Every feature in the feature table has a primary key and a single row of attribute data that is typically a referenced into other tables. Feature tables are joined with the primitive tables on the feature primary key to access the primitives of a feature.

- **Primitives**

There are three geographic primitives, nodes, edges, and faces, and one cartographic primitive text. All geographic primitives can be linked together by topological relationships. Using vector geometry these four primitives can represent any geographic phenomena. Figure 9 shows these primitives. In addition, it can be seen in Figure 9, that there are two types of nodes: entity nodes, and connected nodes.

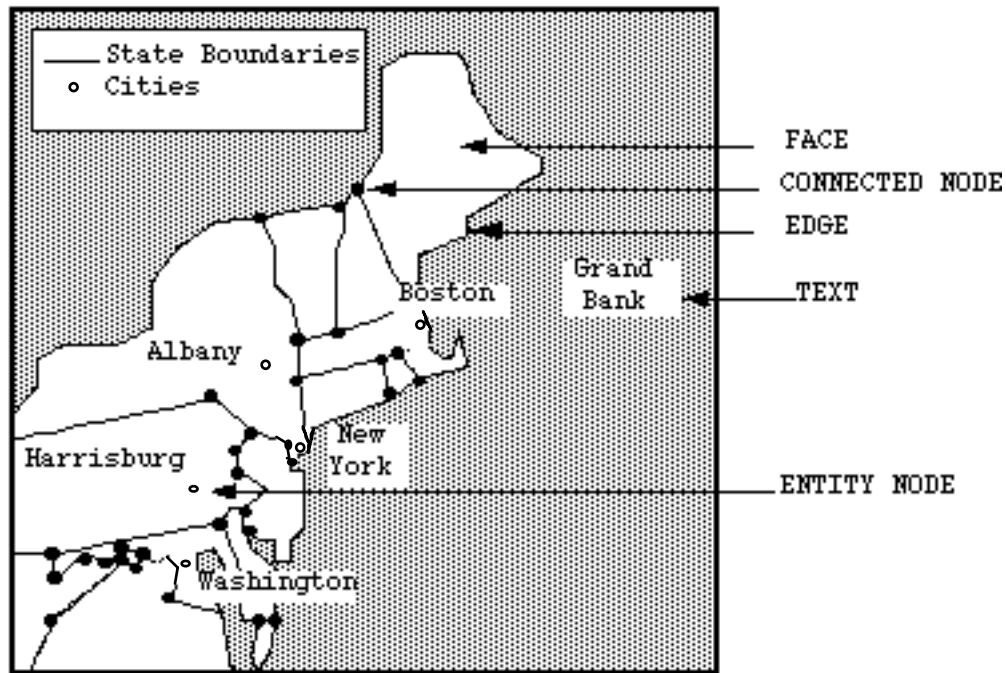


Figure 9. Geographic primitives in VPF (DoD, 1996).

- **Nodes**

Nodes (entity and connected) define a zero-dimensional significant location. There can be only one node for a coordinate tuple. Entity nodes are the nodes that cannot fall on an edge. Nodes, describe isolated primitives such as water towers or primitives which are too small to resolve at the collection scale. Connected nodes are used to define edges by simply specifying the start and the end of the edges, or represent the point features that are located at the start or end of an edge of linear primitives.

- **Edges**

Edges define one-dimensional linear features such as roads, railroads, and boundary lines. Each edge table has an associated edge bounding rectangle (EBR) table to define the MBR of the edges.

- **Faces**

Faces represent two-dimensional area features such as countries, vegetation areas, and lakes. In VPF format, faces are non-overlapping and may have inner rings. This representation allows faces to have inner faces. Each face table has an associated face bounding rectangle (FBR) table to define the MBRs of faces.

- **Text**

Text is not a geographic primitive and has no topological information. The text primitive is used simply to give names to primitives and can be placed at any geographic location.

There are two types of feature classes: simple feature classes (point feature class, line feature class, area feature class, and text feature class) and complex feature classes. A complex feature class is composed of one or more simple or complex feature classes, and a single complex feature table. In total, a feature class consists of several tables that are primitive tables, feature tables, optional join and related attribute tables. In addition, there is a feature class schema table, which describes the relations of the feature class with other primitives. The feature class schema table is used for joining the tables. Figure 10 shows the structural schema of a feature class.

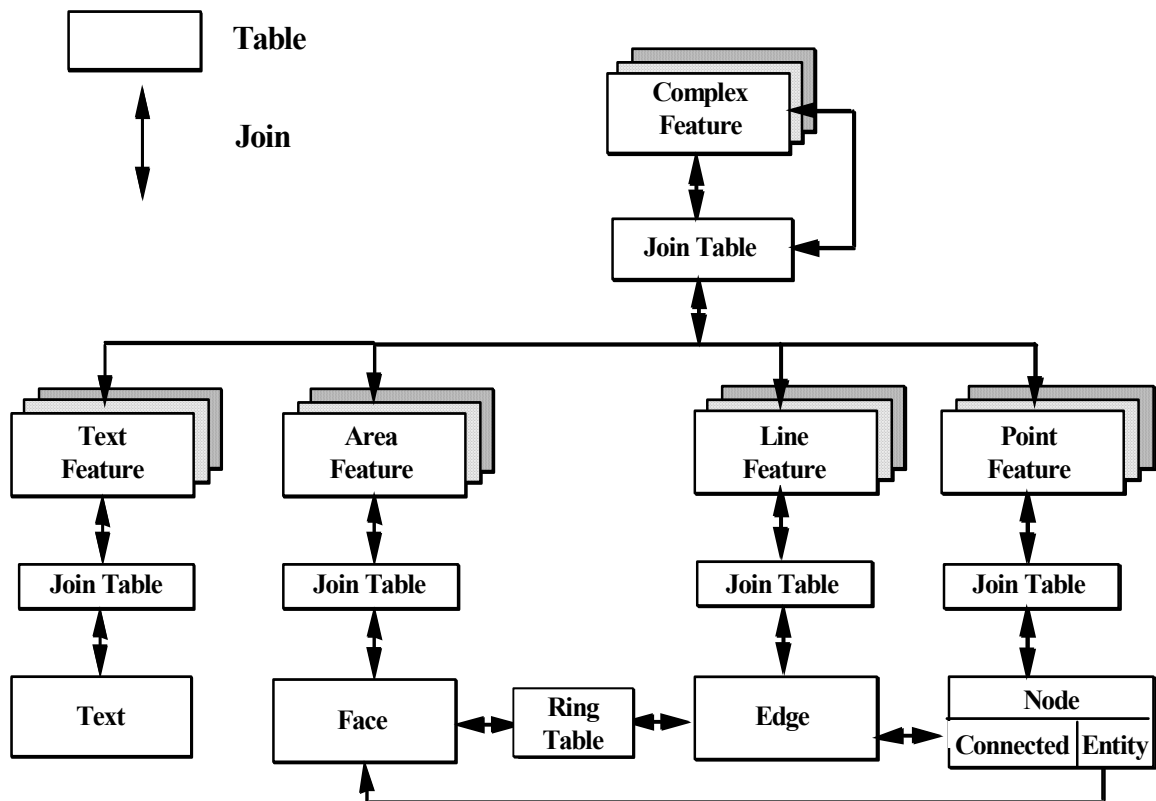
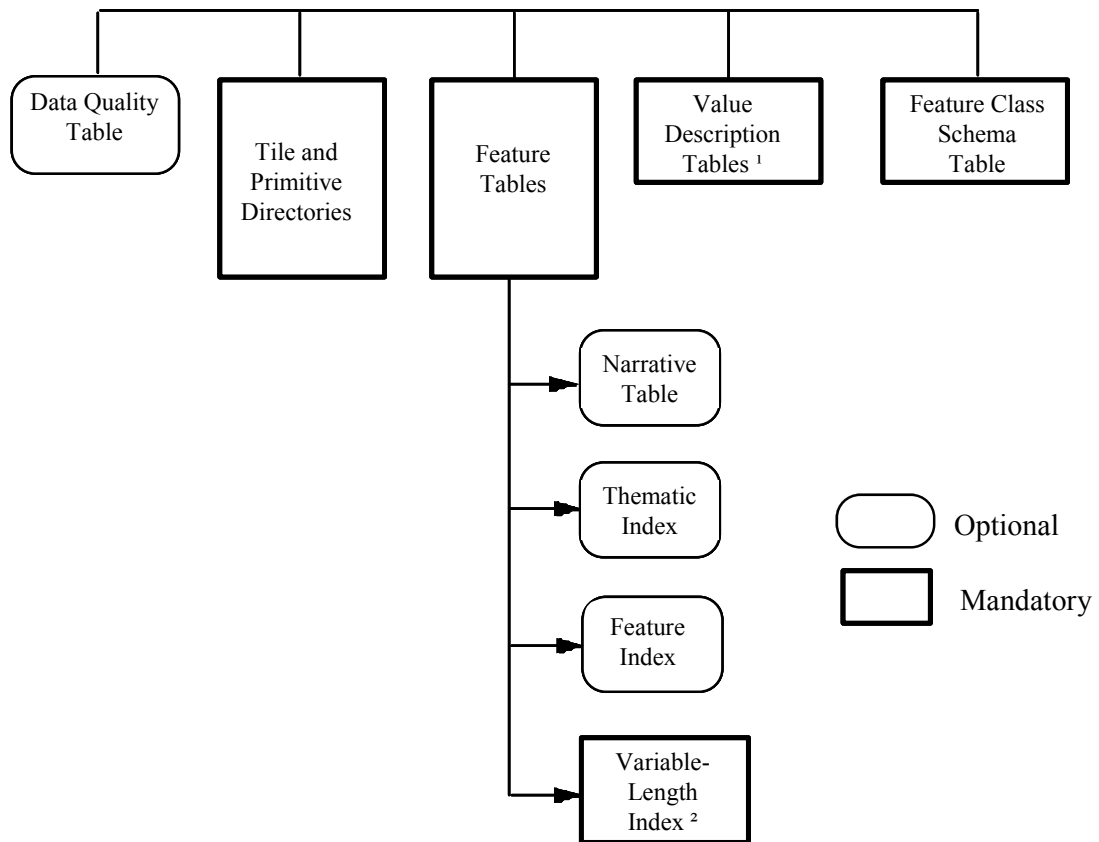


Figure 10. Feature class structural schema (DoD, 1996).

2.3.2.2.2 Coverage

Coverage is a topological combination of features according to topological relationships between the feature primitives. Transportation and elevation are examples of coverage. Every coverage has a level of topology (level 0, 1, 2, or 3). All information regarding coverage is stored under one directory such as “trans” for the transportation grid and is composed of primitive tables, feature tables, and feature class schema table. The structure of a coverage is shown in Figure 11.



1. Mandatory when coded attributes are used
2. Mandatory when variable length column is defined in a table

Figure 11. Coverage class structural schema (DoD, 1996).

- **VPF Topology**

There are four levels of topology in VPF starting at level-0. Level-0 topology implies that there is no explicit topological information and level-3 topology implies that there are explicit topological connections. Nodes topologically define edges and edges topologically define faces. The main purpose of the topology organization is to help with the query and retrieval of features. Figure 12 describes each topological level and gives an example of each level.

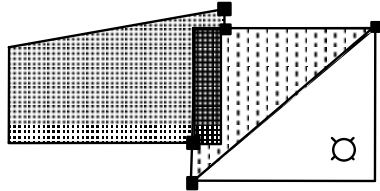
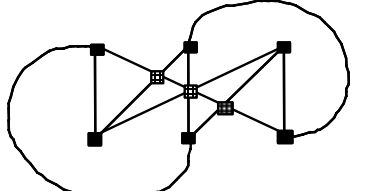
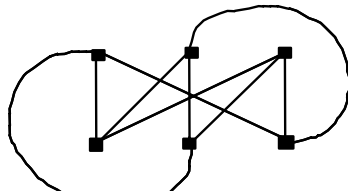
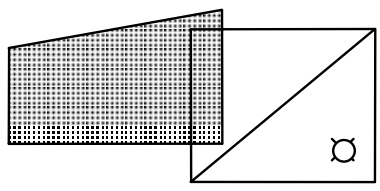
Level	Name	Primitives	Description	Example
3	Full topology	Connected nodes, entity nodes, edges, and faces	The surface is partitioned by a set of mutually exclusive and collectively exhaustive faces. Edges meet only at nodes.	
2	Planar graph	Entity nodes, connected nodes, and edges	A set of edges and nodes where, when projected onto a planar surface, the edges meet only at nodes.	
1	Non-planar graph	Entity nodes, connected nodes, and edges	A set of entity nodes and edges that may meet at nodes.	
0	Boundary representation (spaghetti)	Entity nodes and edges	A set of entity nodes and edges. Edges contain only coordinates, not start and end nodes.	

Figure 12. VPF topological levels (DoD, 1996).

The topology level is carried in edge and node tables. Some columns in the edge and node tables are required for a specific topology. These columns determine connectivity and adjacency. Mandatory columns according to the topology levels are listed in Table 2 for every edge and node table.

Table 2. Mandatory columns for topological levels (DoD, 1996).

Level	Primitive	Mandatory Columns
3	Face	RING_PTR
3	Ring Table	FACE_ID, START_EDGE
3	Edge	START_NODE, END_NODE, RIGHT_FACE, LEFT_FACE, RIGHT_EDGE, LEFT_EDGE
3	Entity Node	CONTAINING_FACE
3-1	Connected Node	FIRST_EDGE
2-1	Edge	START_NODE, END_NODE, RIGHT_EDGE, LEFT_EDGE
2-0	Entity Node	(none)
0	Edge	(none)

- **Tiled Coverages**

To enhance data management, coverages are geographically subdivided into tiles. Tiles are organized as subdirectories under the coverage directory and their purpose is to support the spatial decomposition of features. The logical interpretation of the tiled and untiled coverages is the same. Tiled coverages have the same information as the untiled coverages, the only difference is the primitives are stored in the tile directory according to their tile id. Figure 13 shows geographically subdividing the untiled coverage into tiles.

2.3.2.2.3 Library

The combination of coverages that share a single coordinate system and scale is a library, which has a common thematic definition. Libraries are contained within specified spatial boundaries. All coverage directories and tables must be in the master library directory. If any of the library coverages are tiled, then all of the other coverages must use the same tiling schema or must be untiled.

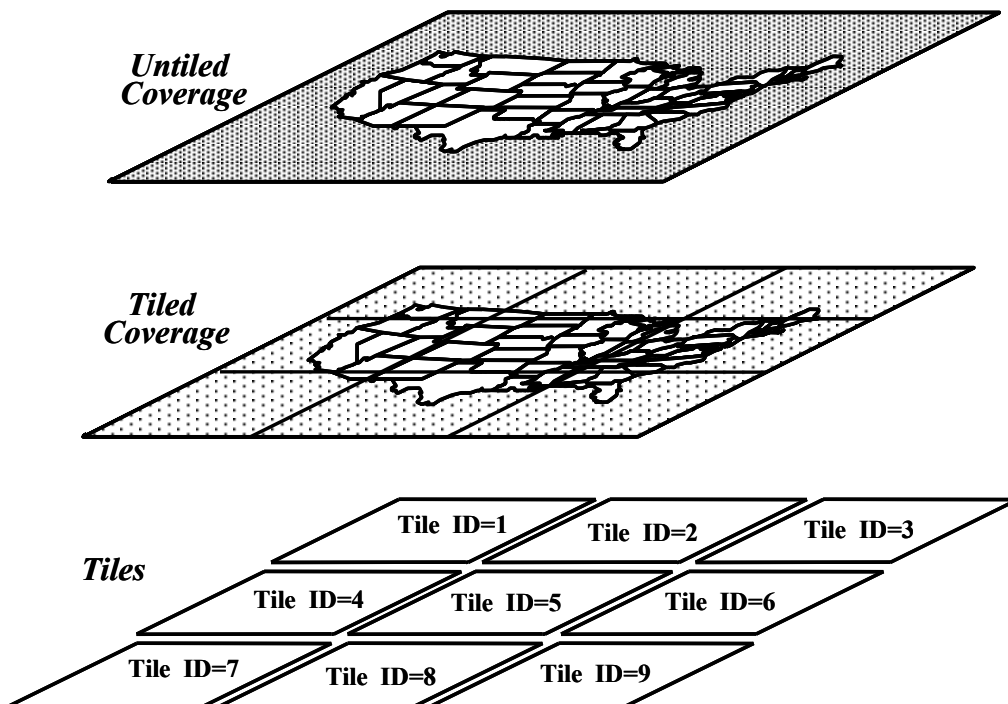


Figure 13. Tiled and untiled coverage (DoD, 1996).

2.3.2.2.4 Database

A database is composed of several libraries. Every database has two mandatory tables: a library attribute table and a database header table. The information in these tables refer to all the data in the database. The library attribute table contains the library names and their MBRs under that database. The database header table defines the database content and security information.

2.3.2.3 Data Quality

The content and the location of the data quality table is product-specific, but it is recommended that at least one table exist at the library level. It contains information about the source of the geographic data. It also contains quality information such as completeness, consistency, date status, attribute accuracy, and positional accuracy.

2.4 Visualizations

Spatial databases are organized to share spatial and non-spatial data, and they are generally application independent. Many different applications may use the same spatial database for different goals. Most of the applications require visualization of the GIS data to some extent. For this reason, most of the vendors also provide the required applications to visualize their GIS data. For example, NIMA (National Imagery and Mapping Agency) provides the VPFView application with their GIS data for free (NIMA, 1996). Some vendors like ESRI (Environmental Systems Research Institute) sell a complete solution kit for their product including spatial data base management and visualization; they also provide some restricted versions for free. In addition to these systems, there are commercial and open source application developers working on the visualization of GIS data. However, because the spatial data format changes from vendor to vendor, these applications are specialized in some areas. In this section, visualizations of the GIS data are examined.

2.4.1 ArcGIS

ESRI has a complete integrated GIS package known as ArcGIS. It has three major parts which are integrated and built on the same structure. Many users can use different applications and/or environments, but can share the same data. The three main parts of ArcGIS are: the ArcGIS Desktop, ArcSDE, and ArcIMS. The ArcGIS Desktop is the complete package for the desktop users. ArcSDE is the complete package for the ESRI's SDE spatial database applications. ArcIMS is a complete package for the Internet based GIS applications. ArcMap is the main component for the visualization of the geo-spatial data and the entire applications are based on the ArcMap component. It has a full functionality for map analysis, editing, and cartography. It supports a layering structure and scaling for the maps. All the other applications such as ArcView, ArcEditor, and ArcInfo use this component as a base structure for the visualization with extra additions. ArcGIS system also has other tools for format conversions to support the interoperability of the GIS (ESRI, 2001).

2.4.2 VPFView

NIMA supports this application for its users, and VPFView is not a complete GIS application. The main purpose of VPFView is to provide limited visualization and query capabilities to users. As can be understood from its name, it is designed for databases that are built on the VPF format. It can reach the database by two means:

- It can directly read the VPF data from the database,
- It defines some extra file structures over the VPF directory/file structure. Generally, this additional file structure is already constructed in the NIMA database in the “view” directory.

The user can select or query features from one or more databases for visualization. The application then creates views or themes for the selection/query. The user can save the results as:

- A graphical display

- Text
- Row data
- A user-defined data structure

The capabilities of the VPFView are somewhat limited. However, it's free and allows the user to combine different databases, and query the database for visualization. NIMA also has some tools contained in the NIMA MUSE package. This package allows the user to change the data formats from raster and VPF formats to other formats, and has a viewer for raster data formats (NIMA, 1996).

2.4.3 OpenMap

Generally, GIS applications are organized in three layers: the user interface (UI), the application, and the spatial database. While OGC specifies the interface between the application and the database layers, OpenMap specifies the interface between the application and UI layers (Cranston, 1999). OpenMap supports various GIS data formats from different vendors. This property of the OpenMap makes it a geo-spatial middleware system (Doyle, 1999).

OpenMap is an open source application, which has been developed by BBN Technologies. It is implemented in Java and all OpenMap components use only the standard Java classes provided with the Java 2 platform. The OpenMap architecture is as shown in the Figure 14. Most of the following information for this section is gathered from BBN Technologies (BBNT, 2001).

The OpenMap architecture is built on three main classes: Projection, MapBean, and Layer. In its layered architecture, OpenMap has a Layer class at the lowest level, which is responsible for reading the data from its source. The MapBean class combines different Layer classes and integrates them as one map. The Projection class projects map data onto the MapBean component. Figure 15 shows the detailed architecture of OpenMap.

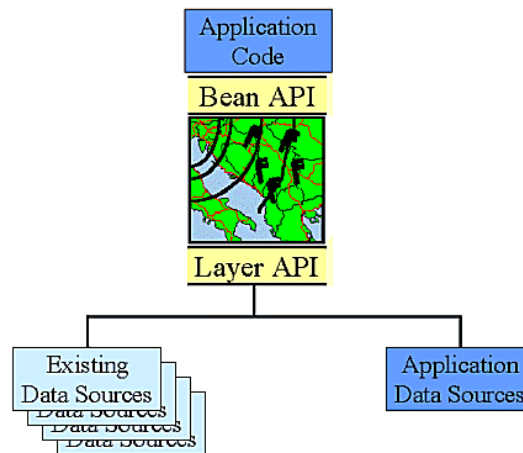


Figure 14. Overall OpenMap architecture (BBNT, 2001).

2.4.3.1 MapBean

MapBean is one of the most important classes of OpenMap. It is simply a canvas that organizes different layers on a hierarchy and allows the Projection components to project the layers. MapBean is derived from the Swing component “JComponent.” Figure 16 shows how MapBean works. Many different layers use the spatial data and all are organized on the MapBean as one-map.

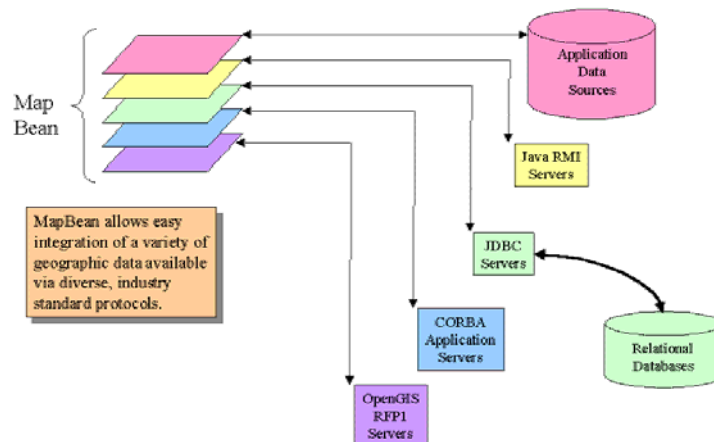


Figure 15. Detailed OpenMap architecture (BBNT, 2001).

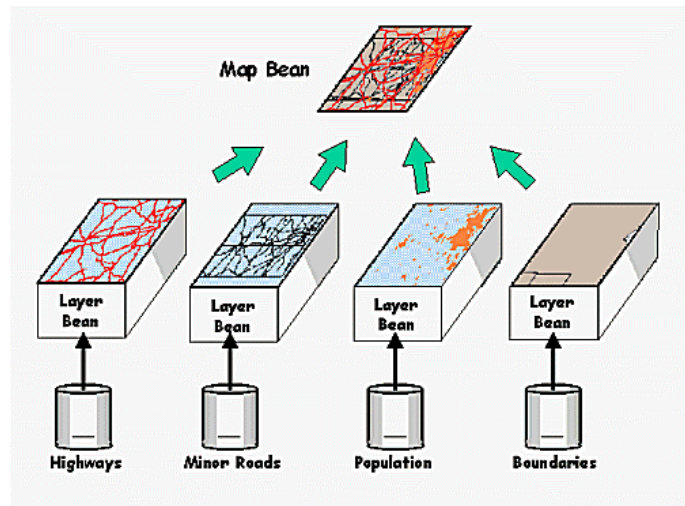


Figure 16. Organizing layers via MapBean (BBNT, 2001).

2.4.3.2 Layers

Layers are responsible for getting spatial data from the databases and displaying the data. Layers are added to MapBean in a hierarchical order. Layers can also register events such as mouse events and projection change events. Layers use the OMGraphics (OpenMap Graphics) component to visualize the spatial data. The OMGraphics components support both vector and raster based spatial databases. OpenMap layers support the following spatial databases or formats.

- VPF (Vector Product Format), DCW (VPF Database), and VMAP (VPF Database)
- CADR (Compressed ADRG)
- CIB (Controlled Image Base)
- RPF (Raster Product Format)
- DTED (Digital Terrain Elevation Data)
- Shape Files
- Arc/Info or GeoTIFF

- ETOPO

2.4.3.3 Projection

After layers gather the spatial information from the databases and render them using OMGraphics components on the MapBean. Projection class then projects the data according to projection type such as Orthographic and Mercator.

2.5 Summary

Most of the research areas in traditional and spatial databases are the same. Management of spatial data however is more complex and costly. For this reason, one of the biggest problems of spatial databases is performance. Spatial database researchers are searching for new ways to handle spatial data while traditional databases have been researching the same problems for non-spatial data for many years. Researchers aim to increase the general performance of spatial databases such as building efficient data models, indexing, and query processing. However, there is also a need to concentrate on other topics of database management systems such as cost estimation and query optimization.

III. Methodology and Design

3.1 Introduction

Geo-spatial databases are used by a wide variety of applications. Many different users query and retrieve geo-spatial data from geo-spatial databases by means of many different applications. The aim of using geo-spatial data is unique to each user. For example, different users can query climate, vegetation, cities, parks, roads, and neighbors of a state. However, few organizations gather data for geo-spatial databases. These organizations must support all different users with their geo-spatial data. To accomplish this, these organizations must organize their geo-spatial data model to capture all information about relations between geo-spatial objects. Such a data model becomes complicated and affects the general performance of geo-spatial databases.

To perform query operations on spatial data, geo-spatial databases require new data types such as nodes and edges to represent the features and their relations, creating complex data models. For this reason, complexity is the most important characteristic of geo-spatial databases. The second most important characteristic of geo-spatial data, large volume, affects both indexing and query processing of geo-spatial databases. Indexing and querying this complex data model must be organized in a way that they support effective and efficient management of geo-spatial data. For these reasons, many research efforts have been undertaken to improve the general performance of geo-spatial databases, especially in indexing. However, their performance cannot be compared with the performance of traditional databases because of the complexity of spatial data and the generalization problem.

Some applications, however, may not need this generalized and complicated structure. For example, flight simulators, mission planning tools, and other simulation tools generally just visualize geo-spatial data, they do not perform any complicated spatial query commands. However, these applications are also affected by the complexity of geo-spatial databases. This research is focused on improving performance of geo-spatial databases for visualization instead of improving performance of geo-spatial

databases for general applications. Applications that just visualize geo-spatial data may not need all the information and relations of geo-spatial objects and geo-spatial data may be organized and indexed in a way that improves the performance of geo-spatial databases for visualization.

3.2 General Approach

While much research is focused on improving the general performance of geo-spatial databases for all applications, this research aims to improve the performance of geo-spatial databases for visualization. For this reason, the first step is to understand the differences between these applications: general applications and applications that just visualize geo-spatial data.

Applications that visualize geo-spatial data mainly read geo-spatial data from geo-spatial databases and visualize this data. The most important thing for these applications is user interaction. The user can navigate on the map by using a mouse or other tools. While interacting with the map, the user should not wait for long queries. Generally, the user wants to see a response from the map immediately, otherwise finding a place from the map can take too long. Also these applications generally visualize geo-spatial data to support their overall purpose. For example, a meteorologist can examine a thunderstorm in his main application by using a visualization tool and if the performance of that visualization tool is poor then this affects the overall performance of main application. The meteorologist cannot wait for long map updates to examine a thunderstorm because the thunderstorm is more important than the map. However, for other applications, the user may query the whole geo-spatial database for specific information without any visual aid. This user can wait for that information because the important thing for the user is the information not the visualization of geo-spatial data. For example, the user may want to learn the neighbors of a country and this user can wait the results of the query because it is the important thing for the user.

Reading Data vs. Altering Data

The visualization objective is simple, just read and display geo-spatial data. Visualization tools generally just read geo-spatial data, they do not add, delete, or alter any geo-spatial data in geo-spatial

databases. For example, a meteorologist generally does not add or delete a road in a geo-spatial database. One property of indexing schemas that are used today is their dynamic nature. However, visualization generally does not need this dynamic nature and the indexing schema of a geo-spatial database may be organized to improve performance for visualization.

Spatial Data

The second property of visualization of geo-spatial databases is to use only geo-spatial data and omit non-spatial data. A visualization tool displays geo-spatial data only. On the other hand, geo-spatial databases have two kinds of data: spatial and non-spatial (attribute data). The data model can be organized to handle just geo-spatial data because visualization does not use non-spatial data. This simplification of the data model enhances performance.

Query Processing

The third property of visualization of geo-spatial data is query processing, which is composed of only one simple spatial query command (rectangle/window) which requires to retrieve features that intersect with a rectangular area that represents the area of the interest. The indexing schema and data model of a general-purpose geo-spatial database is organized to handle complex query commands such as intersect, overlap, and distance. For this reason, a geo-spatial data model is generally in a very complex manner to keep every relation of geographic objects between each other. For example, a connected node knows its related edges and features in a VPF database (DoD, 1996). An edge knows starting and ending nodes, right and left faces, and edges related with itself (DoD, 1996). Figure 17 shows the relation of an edge with other primitives. By combining these primitives, a feature knows every relation of its primitives and rings. This organization may be very helpful for applications that use complex spatial query commands, but for visualization it's overly complex. Visualization just needs one query command, intersection with a rectangular area that represents the display. For this reason, the data model and indexing schema can be optimized to handle this query command only.

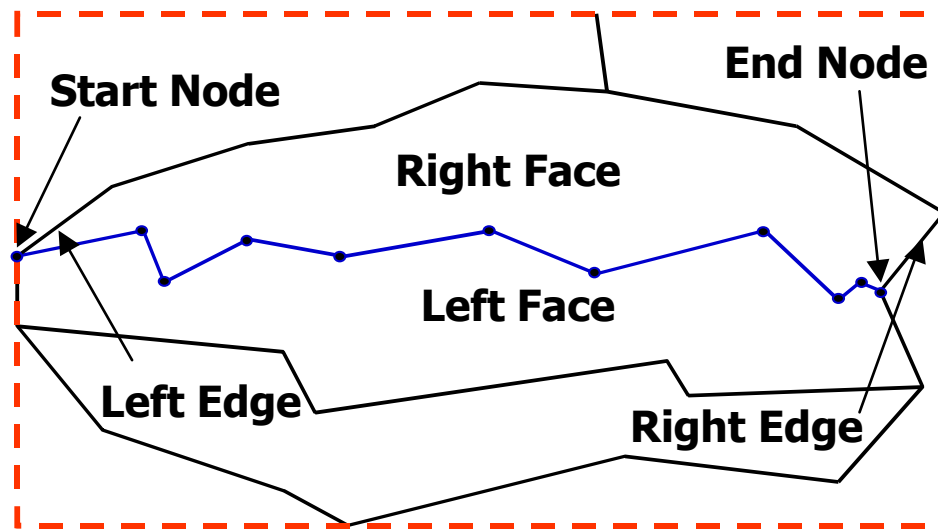


Figure 17. Edge primitive and its relations with other primitives.

Complex Data Models vs. Simple Data Models

A feature or a geographic object may be very complex because of the need to capture all the relations between the features in order to handle complex spatial queries. In geo-spatial databases, features must be indexed because query commands must reach a feature according to its geo-spatial data. This complex feature structure affects the indexing schema because these complex features can be composed of hundreds of lines that cannot be indexed directly. For this reason, approximations are used to represent the features. However, this structure makes searching difficult because searching requires two steps, filtering and refining, as discussed in Chapter 2. The filter and refine technique slows down the searching speed for visualization. These complex features are not needed for visualization.

The question of whether features can be indexed according to their real geo-spatial data (latitude and longitude coordinate values) instead of approximations can be asked. However, without eliminating the complex features of a general geo-spatial database, an indexing schema based on latitude and longitude values of nodes cannot be established. Complex features can be broken down into smaller primitives such as nodes and edges. When this decomposition is performed, an indexing schema that is based on multidimensional indices can be built using the latitude and longitude values of simple primitives. This allows access to each primitive directly by using its latitude and longitude values.

These properties of visualization tools are used to build a new data model and indexing schema that is based on the latitude and longitude value of primitives. This new schema allows accessing every primitive by its latitude and longitude value and this is the only essential query type for visualization. The following sections in this chapter discuss the methodology of building this new data model and indexing schema, and loading it from an old geo-spatial database.

3.2.1 Data Model

In geo-spatial databases, when a rectangle query is processed, every complex feature's approximation value must be checked regardless of the feature's position in relation to the rectangle. To avoid this situation, spatial indexes like R-trees are used but they are still using approximations and they return complex features which may intersect with the rectangle area or may not. The application must read every candidate complex feature into memory and check its coordinate values. This means that even if the only a small part of the candidate feature intersects with the rectangle, all of the candidate feature is read into memory.

As shown in Figure 18, the approximation method eliminates one feature (E), and returns three features (A-B-D) and one non-intersecting feature (C). In this approach, the application must read the entire feature C to memory and compare it with the rectangular area. Although only a little part of the feature D intersects with the rectangle area, the application must also read the entire feature D.

For this reason, instead of a complex data model, a simple one, which is composed of only nodes and edges, was designed. This new data model allows indexing every node according to its coordinate values. All the complex features are broken down into nodes and edges as shown in Figure 19.

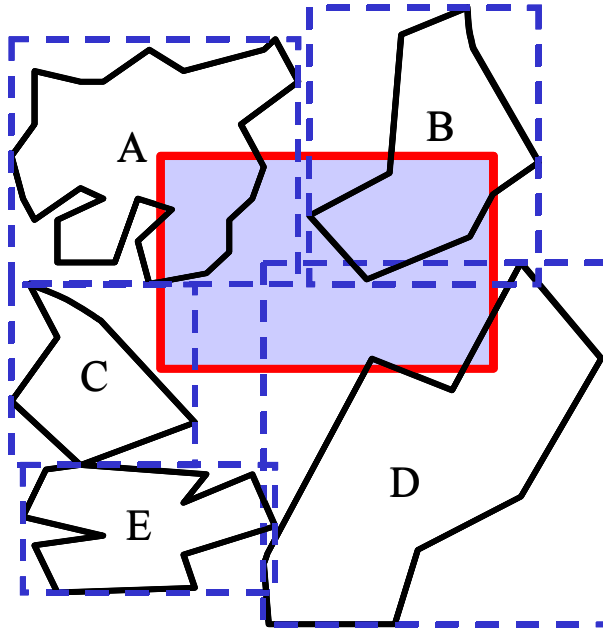


Figure 18. *Querying spatial data via approximations.*

The main primitive of the new data model becomes the node. Every node has an attribute that points to other nodes. If this attribute of a node is null, then this means that the node is an entity node (not connected node). If this attribute points to one or more than one node, then this means that the node forms an edge with that connected node and visa versa. The organization of new data model on disk is a little different. Instead of using pointers to connected nodes, their latitude and longitude values are kept to increase performance. This organization of nodes duplicates some information, but it allows an indexing schema to reach every node by its coordinate values.

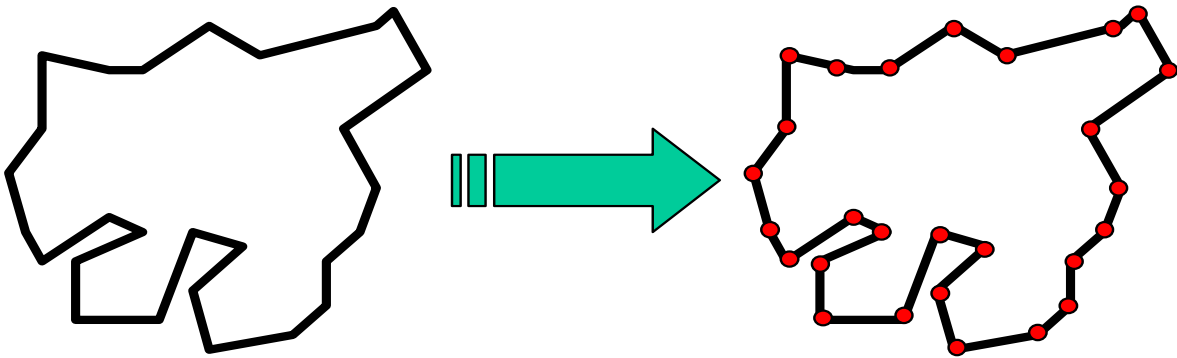


Figure 19. *Eliminating complex features.*

3.2.2 Indexing

The new data model allows building a new indexing schema based on primitive coordinates instead of feature approximations. There are still problems to overcome, however. First, there is a need for a two-dimensional indexing schema. Second, one of the properties of spatial data, distribution of objects in space, makes it difficult to use real coordinate values for indexing.

Dividing space into smaller cells is used to solve the second problem. This is one of the general methods that is used in spatial databases. However, in this indexing schema instead of using big cells, small cells are used to represent the coordinate values of the primitives. A cell may contain more than one node that forms a neighborhood relation. Making the cells very small decreases number of nodes in a cell and increases the indexing performance. These cells are indexed to increase data retrieval performance.

The two-dimensional indexing problem can be solved with a two-dimensional matrix approach that maps every coordinate value to a matrix cell, but the distribution of objects again makes this approach almost impossible. The number of the empty cells cannot be calculated or estimated, and most of the matrix cells may be empty. Having large amount of empty cells in the matrix structure decreases efficiency significantly. For this reason, the space is divided into rows (latitude cells), and every cell (longitude cell) in a row is organized and indexed with B+Tree structure. Empty cells are omitted. This organization makes the indexing schema effective and efficient. An array is used to point to the root of the related B+Tree structure for every row (latitude cell) in database boundaries as shown in Figure 20.

For a given coordinate, related cell values are calculated from latitude and longitude values. The longitude coordinate value points to a cell in the array structure that points to the root of a B⁺-Tree, which holds all the latitude cells in that row (longitude cell). The B⁺-Tree is searched to locate the correct cell according to the latitude coordinate value. This cell points to its nodes in the node file.

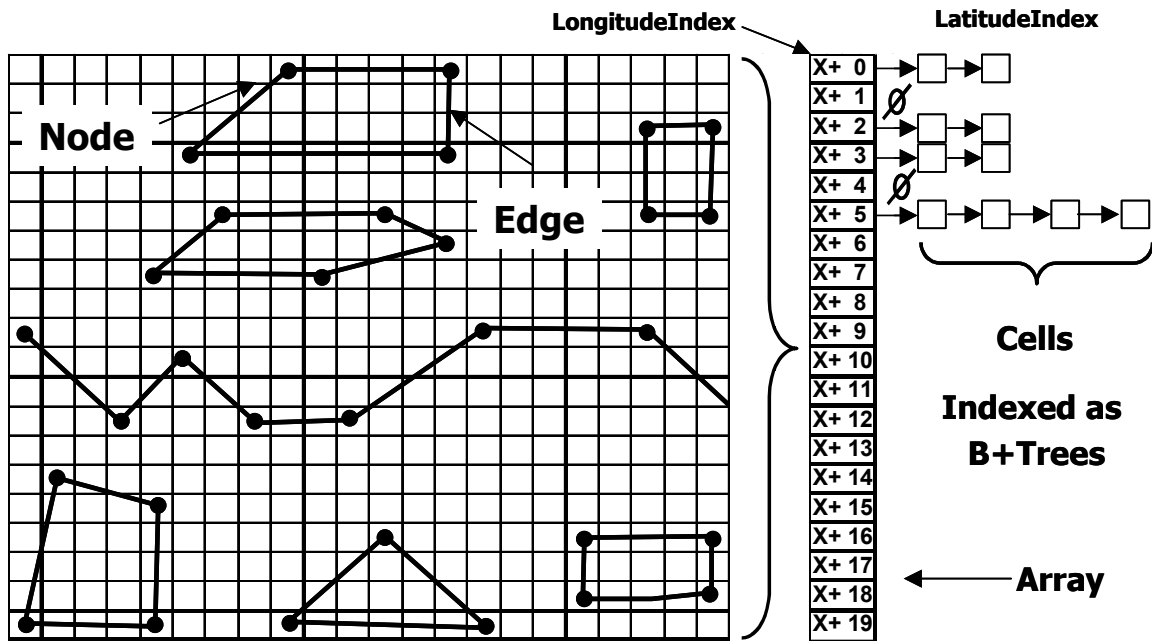


Figure 20. New Indexing Schema.

This organization maps to four files on disk. One file holds the array of longitude cells, one file holds the B⁺-Tree structures for every cell row, one file holds the information of a cell's latitude and longitude key values and a pointer to its first node in the node file, and one file holds the nodes (real data). These files are built for every feature inside the digital map such as roads, rivers, and railroads. The indexing schema writes the cells and nodes to the files based on their latitude and longitude position. By writing the data in the same latitude from west to east, the schema clusters the information in the files. This clustering enables direct access to eastern neighboring cells in a region without requiring multiple reads from the same B⁺-Tree. Hence, visualization applications can go directly to the point of interest without any prior calculation or checking the approximations of features, increasing the performance of the visualizations.

3.2.3 Conversion

A data conversion from an old digital database is performed because there is a need for actual data to test this data model and indexing schema. National Imagery and Mapping Agency's (NIMA) VMAP

database that is created in VPF format (DoD, 1996) is used to load the new database. Because the VMAP database fully supports geo-spatial databases, it is composed of complex features. These complex features are broken down into nodes and indexed for the new database. In the new database, the same file hierarchy of the VPF database is used for additional spatial decomposition such as “database name\library name\coverage name\feature files” (see Figure 8). This means that every “road” primitive is kept inside the same file under the “trans” coverage directory. Because OpenMap already has a layer to visualize VPF data, some OpenMap VPF Layer classes are used for reading VPF data from its file structure. The conversion focuses on the correctness and automation of the process, and is not focused on its speed. The aim is to convert the data without user input.

3.2.4 Visualization

When the data are obtained, it must be visualized to test the data model and indexing schema. For this reason, OpenMap’s libraries are used to simplify the visualization process. An entire layer is implemented under OpenMap structure as GeoSVis. This layer has all the properties of OpenMap layers and behaves as one of them. It reads data from the new data model and displays and projects it using OpenMap libraries.

3.3 Evaluation

After everything is implemented, the next step is focused on testing the performance of the new database. The new system must be tested against a known system. OpenMap was used for the testing process since it has the ability to read and visualize VPF data. Because VPF is the old database that the conversion was made from, a fair comparison can be made. The old and new databases were queried for the same rectangle area with the same map design on the same computer and the test results were compared.

The new system has some advantages other than the performance. The first is platform independence. The new tool is implemented in the Java programming language using binary random access files. This implementation gives platform independence to both the tool and its data.

3.4 Summary

Because the main goal of this research is to build high-performance indexing schema over geo-spatial databases for visualization applications, the current complex data model, approximation approach, and filter and refine techniques of the geo-spatial databases are not used in the new database. Instead, the complex features are broken down into their simple primitives to get rid of the complexity and two-dimensional high-performance indexing structures are used to index these simple primitives. The indexing schema uses the coordinate values of the primitives for indexing instead of the approximations. To obtain high-performance, the indexing schema uses a combination of an array structure and B⁺-Tree structures to index.

IV. Implementation

4.1 Introduction

This chapter discusses the implementation of the data model, conversion tool, and visualization tool. The conversion tool is used to get data from a VPF database, create the new data model, and build a new database. The visualization tool is used to test the performance of the new database and demonstrate that it works.

Because one of the main goals of this research is to create a platform-independent geo-spatial database and a tool to display geo-spatial data from this new geo-spatial database, an enterprise DBMS is not used for the implementation. The entire metadata and geo-spatial data are stored in binary files by using Java 2™ Input/Output classes. The indexing structure is based on a B⁺-Tree data structure discussed later. In addition, all visualization code is also completely implemented by using Java 2 under an OpenMap structure.

4.2 Proposed Data Model

In this section, the proposed data model is examined in more detail.

The main goal of the proposed data model is to get rid of the complexity of the current geo-spatial data model and represent features as strongly as possible. By using this simplicity, the second goal is to build an indexing structure that is based on geo-spatial data itself and not approximations, which allows query processing in one step. These data model and indexing schema must be very basic because a complex structure can affect other applications that will use this structure.

This new data model is based on VPF database structures because VPF data is used to load the new database. For this reason, for other geo-spatial databases, which have additional or different primitive types than VPF primitive types such as “arc”, may require some modifications.

To achieve the first goal of simplicity, there are only two geographic primitives in the new data model: node and connected node, and one topographic primitive: text. Because this new data model is going to be used by only visualization tools, these three primitives are sufficient to visualize all features. All complex features are broken down into these primitives except for node features. For example, edges and faces are broken into series of connected nodes. In addition, the new text feature has only one coordinate instead of having multiple coordinates (note: OpenMap structure supports only one coordinate). The class diagram of the new data model is shown in Figure 21.

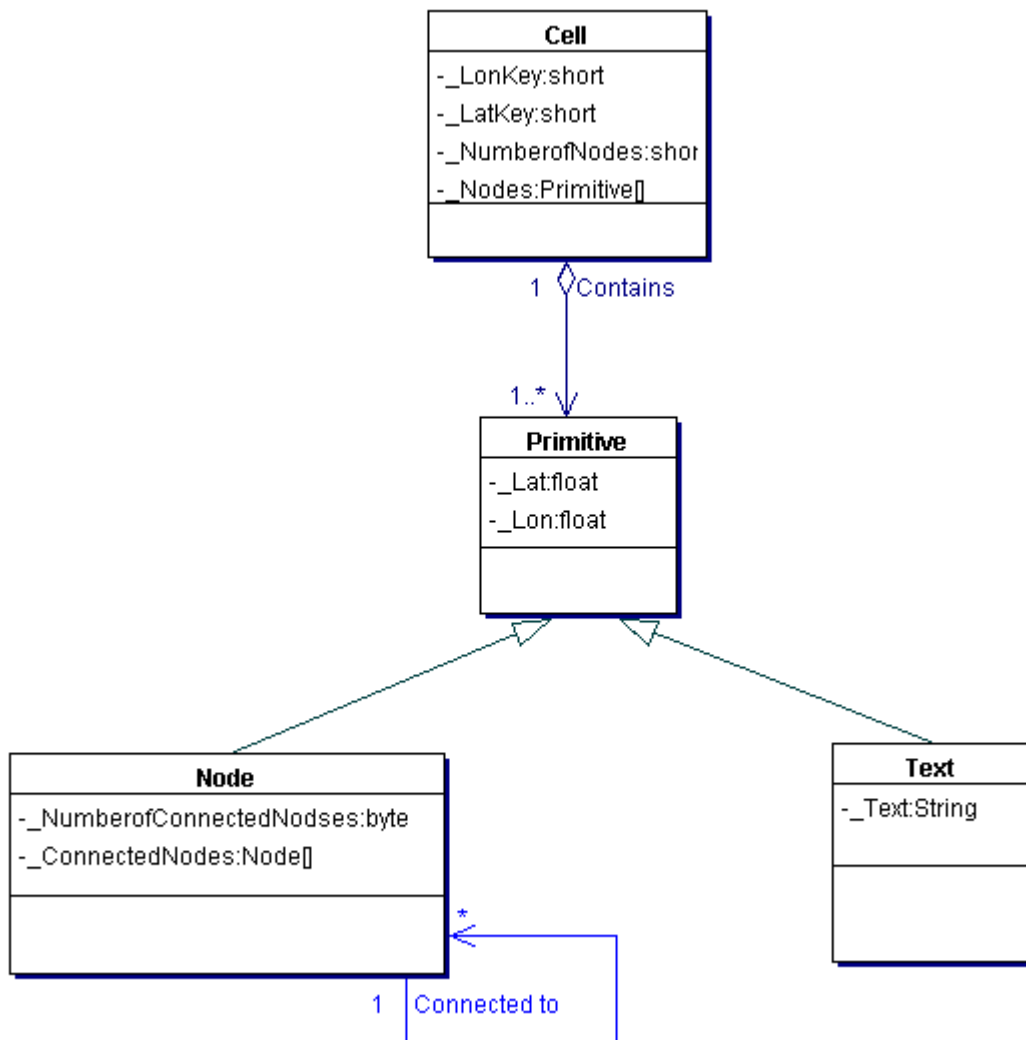


Figure 21. Proposed Data Model.

As can be seen in Figure 21, this data model represents organization of primitives in memory. However, the organization of the data model in the file system is very different and it is discussed in the following sections in more detail. In addition, the file and directory hierarchy of the VPF database is also used for spatial decomposition and it is also discussed in the following sections. Properties of this data model can be summarized as:

- The real latitude and longitude coordinate values of a primitive cannot be used for indexing directly because of the large volume of geo-spatial data and its distribution in space. For this reason, there is a need to organize primitives in space, and the Cell class is used for this purpose. It is not a geographic primitive, it is only used to index primitives according to their latitude and longitude coordinate values. The data space is divided into grids or cells and these cells are used for indexing the primitives inside these cells.
- This data model does not differentiate between nodes and connected nodes. If a node is connected (points to at least one node) then it is used to represent edges and faces. If it is not connected (does not point to any node) then it is a node that is used to represent points. A text primitive is similar to a node; with the addition that it has a string type to hold text value. This allows latitude and longitude coordinate values of any feature to be used for indexing.
- A feature is generally composed of only one type of primitive such as node or text. Every feature has a type that represents its geographic properties such as point, line, area, and text. This organization can also represent a mixed type of features.
- There are four classes above these primitives in the file hierarchy: Database, Library, Coverage, and Feature as shown in Figure 22. These classes organize the database and provide a simple spatial decomposition. Database, Library, and Coverage corresponds to a directory in VPF and in the new database file hierarchy. Feature corresponds to real geographic objects such as roads and railroads and is represented as unique files in the new data model.

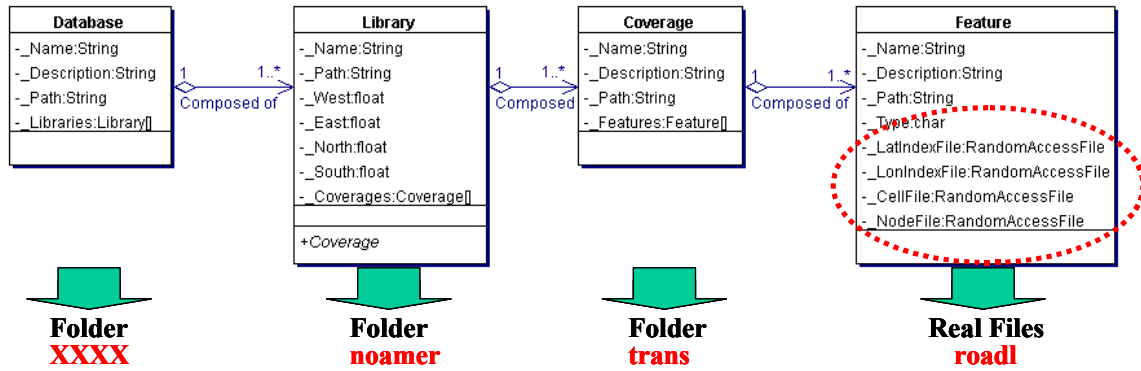


Figure 22. File Hierarchy.

- There is no direct physical connection between the file hierarchy and data model because every cell is immediately converted to a real OpenMap graphic such as line or text after it is read. Making a direct connection means doubling the data in memory in the OpenMap architecture. The file organization is discussed in the following sections in more detail.

- Because this data model is not dynamic all connections between primitives are represented as arrays to increase performance.

A representation of this data model is shown in Figure 23. On the left there is an edge feature and on the right are point features. The data model has the same approach as shown in this example. The data space is divided into cells like a matrix structure. Edges are divided into nodes and every node knows its connected node. Nodes are connected to each other bidirectionally. For example, node Y is connected to node X and node X is connected to node Y.

Logically, the structure seems like a matrix structure, however in the real data model every row of the matrix is captured by a separate B⁺-Tree and only cells that have nodes are indexed in trees. When there is a need to reach a specific node, the entire cell that contains the node is read to memory.

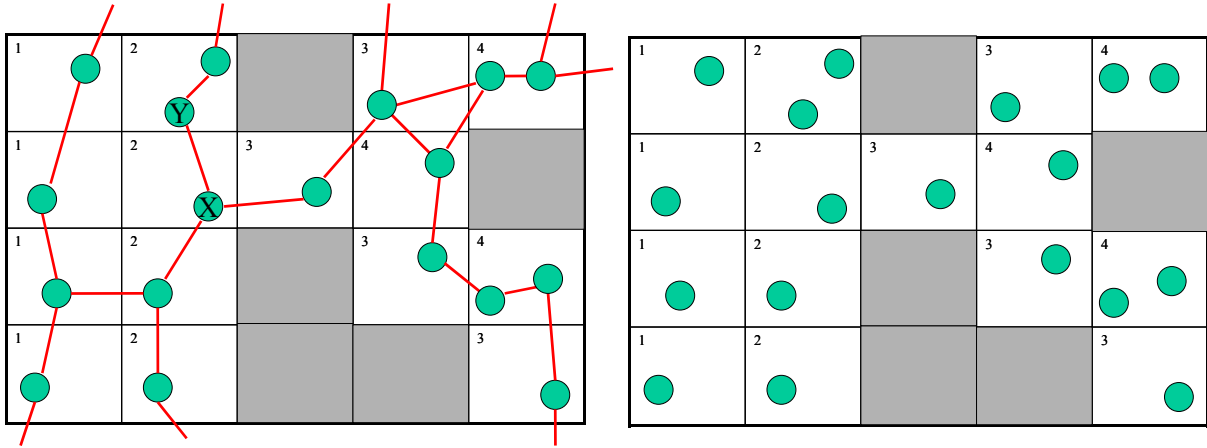


Figure 23. Representation of Data Model.

4.2.1 Visibility Problem

The main primitive of the data model is the node. Every connection between nodes is kept in the nodes themselves. There is no real edge, area, or face primitive in the data model. For this reason, for an edge or area feature at least one of the nodes must be inside the query window, otherwise the edge or area feature cannot be. For example, the dark line between two dark nodes cannot be seen inside the viewing window in Figure 24 because none of the nodes are inside the viewing window. This can be a problem at small scales. However, lines must be very long to cause this problem and most of the real world line features are not long enough to cause this effect. In addition, the probability of this problem occurring is very low because entire cells are read into memory, not individual nodes. Although the cell size may vary, the current cell size is 10NM in both directions and reduces the probability.

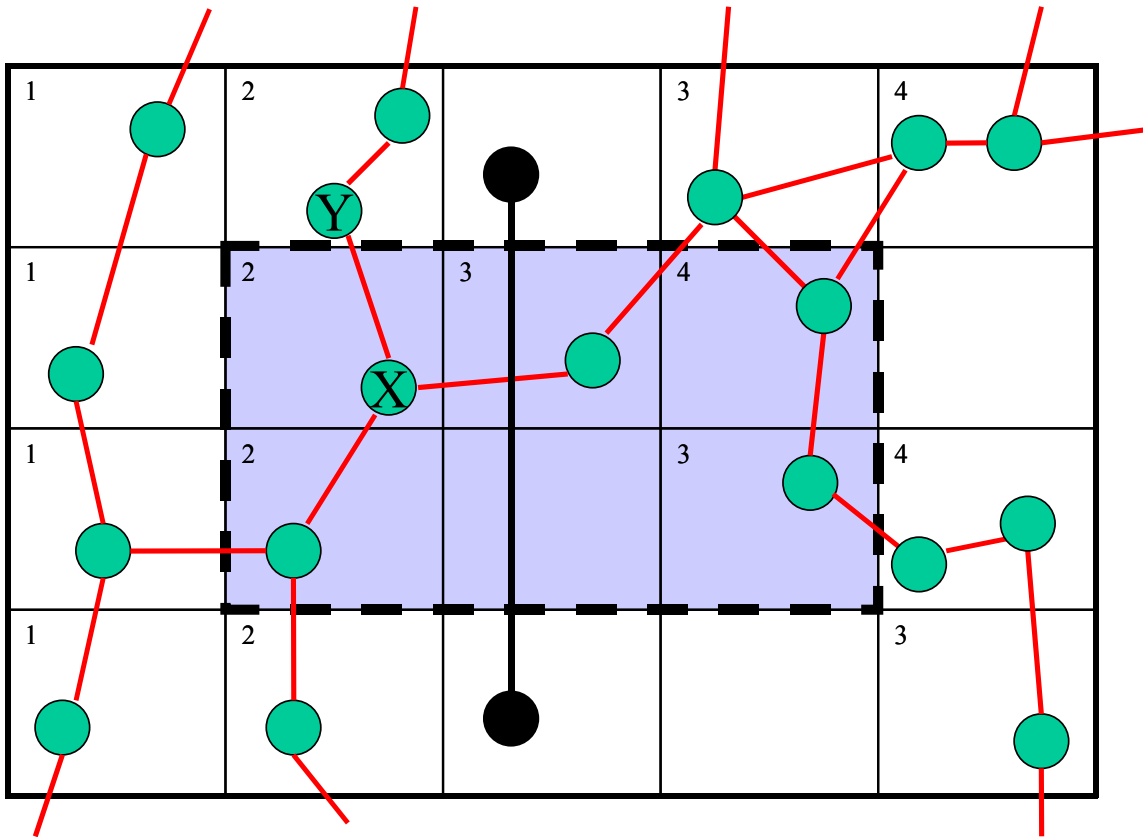


Figure 24. Visibility Problem.

4.2.2 Solving Visibility Problem

The visibility problem may occur at small scales for long line features when any of the nodes are not inside the viewing window. It is important to determine a line size that can probably cause this problem. There are some calculations that can be performed to decide the critical line size. Screen resolution for the visualization program is 800x600 pixels. The line feature must cross the cell boundaries to cause this problem, because the data is read cell-by-cell. For this reason, the small number in the screen resolution (600 pixels) is assumed as the critical line size. However, lines crossing at an angle can also cause this problem. For this reason, the assumption for the critical line size is narrowed to 300 pixels to be more conservative. This critical line size constant is a limit for the long features in pixel unit according to screen resolution. However, to use this constant on the feature data, it must be converted to the units such as NM or kilometers. For this conversion, there is a constant, which is used for projection, to draw

geographic primitives correctly, as pixels per meter in OpenMap architecture. This constant was calculated using a Sun monitor to know the conversion factor from meter unit to pixel unit and its value is 3272 pixels per meter. This conversion can be done based on a projection-scaling factor using assumed critical line size, pixels per meter constant, and scaling factor together. Because the VPF data is collected at 1/1,000,000 scaling, it helps to decide a scaling factor for the critical line size. However, applications may want to use more detailed visualizations using low scaling factors. For this reason, different critical line size values are calculated for different scaling factors as shown in the Table 3 to make a more conservative decision. Although, the VPF data is populated from 1 / 1,000,000 scaling, the critical line size based on this scaling factor is too long for the features (see Table 3). To allow applications to use more detailed visualizations in low scaling factors, 9 KM is decided as a critical line size for the data model. Once the critical size of the line features that can cause the visibility problem is decided, the following suitable approaches to solve this problem may be examined.

1. A minimum-bounding rectangle can hold for every line that is greater than the critical line size. After searching proposed index structure, this minimum-bounding table can be searched to find these critical lines.
2. Every cell can be related with the critical line that passes over the cell.
3. Every edge or face primitive that is longer than critical line size can be divided into equal length segments during conversion.

Table 3. Critical Line Size for Different Scales.

Scaling Factor	Critical Line Size
1/1,000,000	91.687 KM
1/250,000	22.92175 KM
1/100,000	9.1687 KM

One of the main concerns of these approaches is the performance drop caused by the extra work to solve the problem. However, observation of actual NIMA data reveals that the percentage of critical lines is under 3% and does not affect performance significantly. The last approach was selected to solve the problem because it does not add extra complexity to the indexing structure and data model. Dividing is done during conversion and only increases feature and indexing file sizes, and does not require any additional processing while querying database. The other two approaches require additional structures or processing to solve the problem. Figure 25 shows how the visibility problem is solved. Even the database was created with the cell size equal to 1 NM in each dimension, final implementation uses 10 NM. The circled line feature on the upper left corner of the viewing area is a straight line and its actual nodes are out of the viewing area. However, its long line size that passes the cell boundaries are determined during the conversion process and it is divided into equal segments which are smaller than the critical line size. This approach forced the data model to capture at least one segment of the line feature in the viewing area without disturbing its shape.

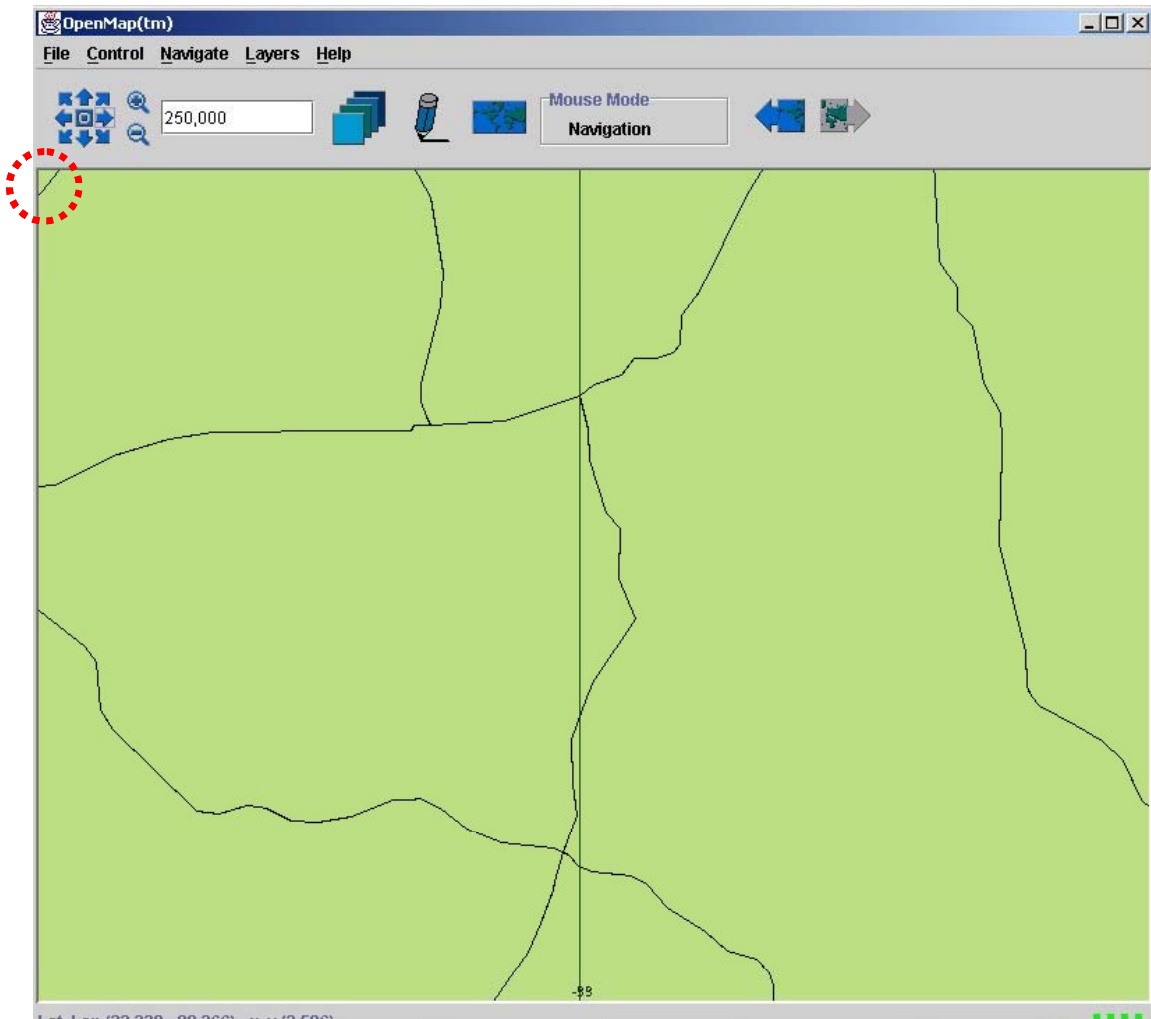


Figure 25. Solving Visibility Problem.

4.3 Indexing Schema

The main purpose of the indexing schema is to index primitives according to their coordinate values. Because the main primitive is the node and all the other primitives can be indexed like node primitives, any two dimensional indexing structure can be used for indexing. However, the large data volume requires storing the entire indexing structure in files. The indexing structure must handle large data sets, and must be very fast.

The data space can be thought of as a matrix structure for a two dimensional data. However, a matrix structure cannot be used to efficiently index this data model because there are numerous empty cells in the data space and there is no pattern to their distribution in the data space. However, when one dimension of the matrix (row or column) is considered, there is a high probability that almost all of the rows or columns will have at least one unempty cell. For this reason, an array structure can be used to organize one dimension and can point into another indexing structure for the other dimension. The array structure can have empty cells but this cost is low since the array has only one long pointer for each row or column of the matrix. The probability of having no cells for any one dimension (row or column) is very low. Every VPF library has a “library attribute file” that determines the boundaries of the library so the array structure can be created very easily by using these values and can be directly accessed by using cell coordinate values without any search.

After deciding the array data structure to use to index one-dimension, now there is a need to use a more efficient indexing structure for the other dimension. This second data structure cannot use direct accessing because of the empty cells, however it must be fast and work on the file system. The B⁺-Tree data structure offers a solution for this problem (Cormen, 2000). It is suitable for indexing the second dimension of the coordinate data.

The latitude coordinate value of a primitive is held by the array data structure and the longitude coordinate value of the primitive is held by B⁺-Tree structure. The B⁺-Trees point to cell files according to the index value and the cell file points to a node file for the primitive data. This structure requires three files for indexing and one file for the data.

Figure 26 shows the indexing structure and how it maps cells in data space. Every element of the array structure represents a row of the data space and points to a B⁺-Tree indexing structure. The B⁺-Tree indexing structure represents cells of that row and points to cell file, which also points to node file for the real data.

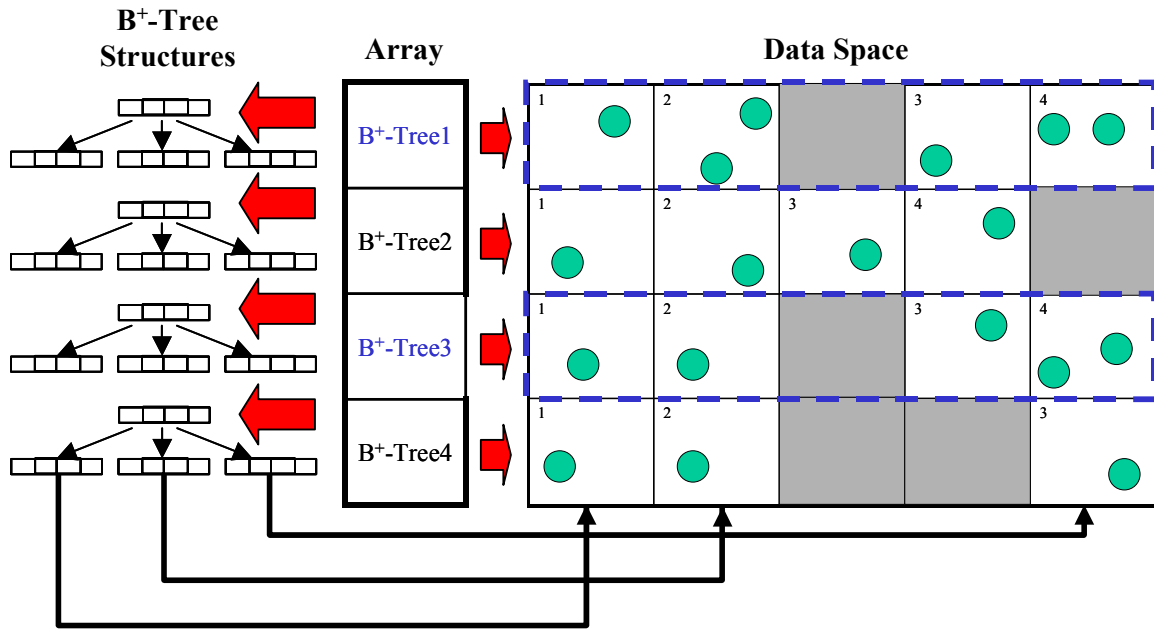


Figure 26. Proposed Indexing Structure.

As mentioned before, there is a file hierarchy above this indexing structure and its purpose is to help spatial decomposition. This file hierarchy divides the database into libraries that are kept in different directories under the database directory. In addition, it also divides libraries into coverages in different directories. Every feature's indexing and data files are organized according to feature name. This file hierarchy is the same as the VPF database. There is a library attribute table “lat” under the database directory and a coverage attribute table “cat” under each library directory, and these files contain information about the entire file hierarchy. These files are used to build the representation of the file hierarchy in memory as shown in the Figure 27.

Under this file hierarchy, there are four files that contain data and metadata of features. These files have their name of the features as file name and filename extensions determined by their data type. For example, the latitude index file has a “.lat” extension, the longitude index file has a “.lon” extension, the cell file has a “.cll” extension, and the node file has a “.nod” extension. Their organization in memory is discussed in Section 4.2 and their organization in the file system is shown in Figure 28.

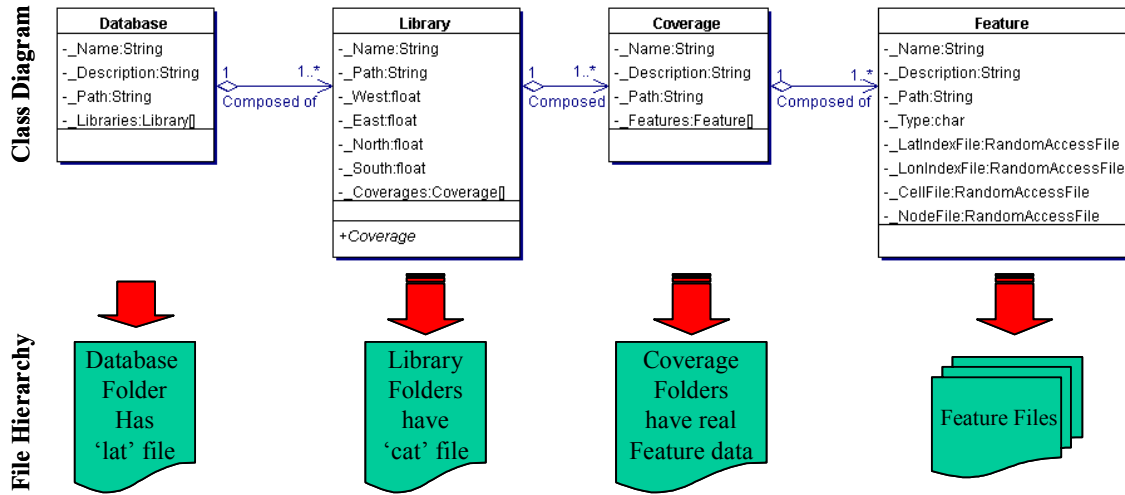


Figure 27. File Hierarchy.

Because there are four files, a cell's data can be read in four steps. First, using a cell's latitude as a key value, a pointer to a related B^+ -Tree can be obtained directly from the array structure without any search. Because this array structure has a constant length and is small, it can be easily kept in memory during the query process. The array structure is the only metadata that can be kept in memory. The related B^+ -Tree's root node is read into memory and it is searched using the cell's longitude as a key value. This search returns a pointer to a cell in the cell file. From the cell file, a pointer to its nodes in the node file is obtained. Every node is read from its node file using its data structure according to its feature type such as point, line, or text. The entire cell data is read into memory from the node file at one time. The node data of the Cells is saved from West to East in the visualization order so there is no need to reach every cell from the indexing structure. Only the first cell is searched via the indexing schema and then all remaining data can be read from the node file. There are some variables that can be changed in this structure such as cell size, branching factor of the B^+ -Tree structure, organization and order of the cell and node files, organization of the array and B^+ -Tree structures, and also some alternative indexing techniques. The approach to evaluating and setting these variables discussed in the following section.

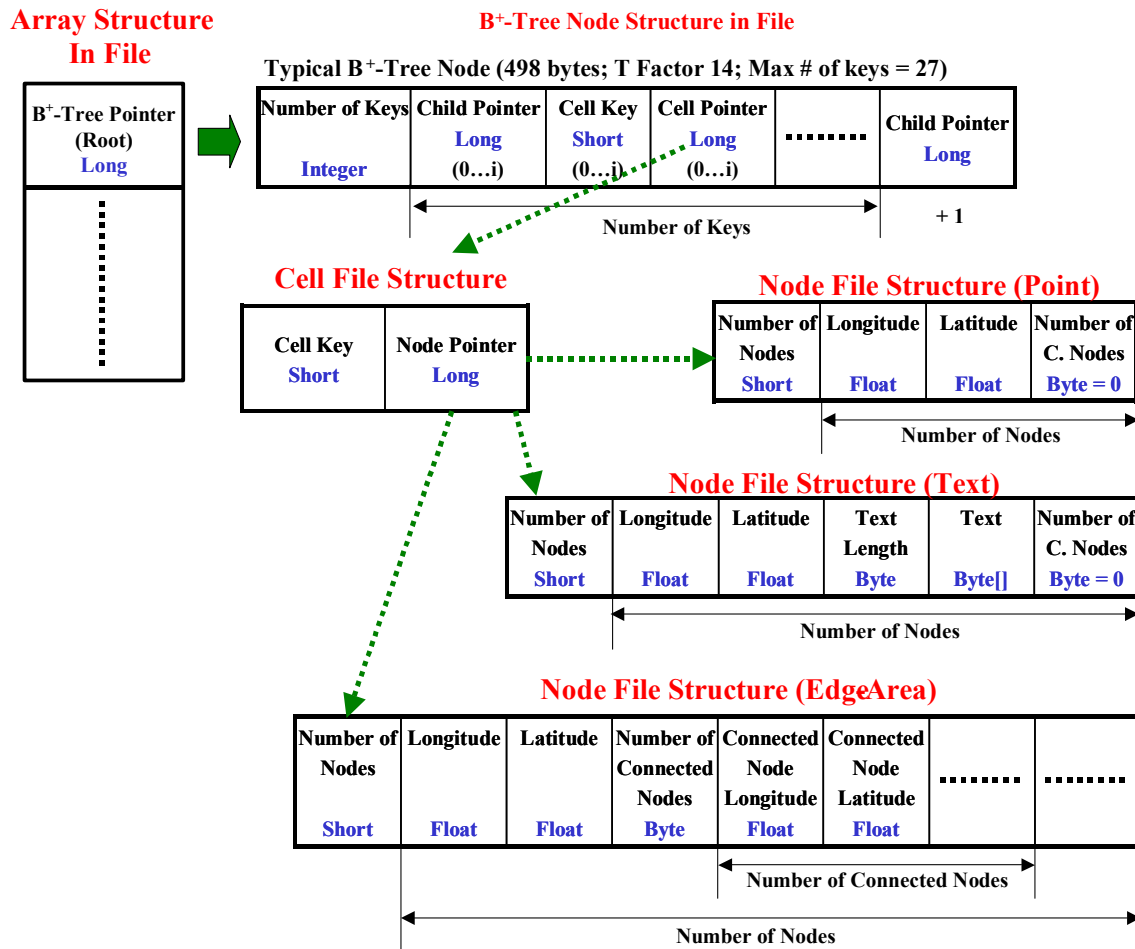


Figure 28. Feature Data and Metadata File Structures.

4.4 Improving The Data Model and Indexing Schema

During the implementation phase many different variations were tested and the data model and indexing schema were modified according to the test results. This section discusses these improvements and the tests that were made during the implementation phase.

Branching factor of B⁺-Tree structure for each feature is the same and it is 27 (T=14). In addition, cell size is equal to 1 NM² area. That structure requires one B⁺-Tree process (creation and search) for each row of the viewing area.

The test results show that the indexing structure reaches the first cell (left upper cell) very quickly, however the overall performance for the entire viewing area is almost the same as a VPF database. Test revealed that the aim of the indexing schema should not only be to reach a specific data item faster; it must also be able to reach the entire data set inside the viewing window faster. However, the four-file indexing structure has inherent problems that prevent it from performing faster. One of the reasons is the large number of B⁺-Tree structures that are required for this indexing structure. For North America alone, almost 18,000 B⁺-Trees are required. The overhead of the large number of B⁺-Tree structures decreases the performance significantly and the following improvements were made to increase the overall performance of the indexing schema.

- **Eliminating The Cell File**

The cell object is designed to index primitives easily. However, there may not be a need for an actual cell file in the file hierarchy because B⁺-Tree keys can point to a node file directly instead of pointing to a cell file. This eliminates the cell file from the indexing structure with only minor changes, and decreases the complexity while increasing performance.

- **Cell Size**

Eliminating cell file improves performance, but not by a significant amount. The real goal must be to improve the overall performance. In the proposed data model and indexing structure, there are two ways to improve the overall performance: the first is to decrease the number of B⁺-Tree structures and second is to decrease the number of B⁺-Tree accesses required for entire viewing area.

One of the easiest ways to decrease the number of B⁺-Tree structures is to increase the cell size. The first selected cell size, 1 NM² area, was designed to reach a specific primitive quickly however it does not help much for an overall speed up. Increasing cell size decreases the number of B⁺-Tree structures, but it also increases the amount of data that is read for a cell. Test results for different cell size alternatives show that increasing cell excessively results in poor performance. After testing, the cell size was set at a 10

NM² area, which decreases the number of B⁺-Tree structures from 18,000 to 1,800. This means less B⁺-Tree accesses according to the previous cell size for the same viewing area.

- **Alternative Approach (One Large B⁺-Tree Structure)**

An alternate approach is to use one large B⁺-Tree structure instead of having an array structure and many B⁺-Tree structures. This structure must map two dimensional cell keys (latitude-longitude) to one key. This can be easily done by combining two keys together since there is only one unique cell for a pair of keys (latitude-longitude) as described in Figure 29. This simple mapping allows separating two-dimensional cells from each other with just one key. The final key can be used in one B⁺-Tree to index cells.

Every B⁺-Tree structure requires reading the root node to create it in memory and searching for a particular cell key. Because the previous indexing schema has many B⁺-Tree structures, it requires creating many B⁺-Trees. On the other hand, indexing the same amount of data in one B⁺-Tree instead of in multiple B⁺-Trees increases the depth of the B⁺-Tree and it means that there are more disk reads required for searching. Test results show that the alternative approach, having one B⁺-Tree structure, is worse than the old approach: multiple B⁺-Trees with cell size equal to 10 NM² area. For this reason, this approach was not used for the final indexing schema.

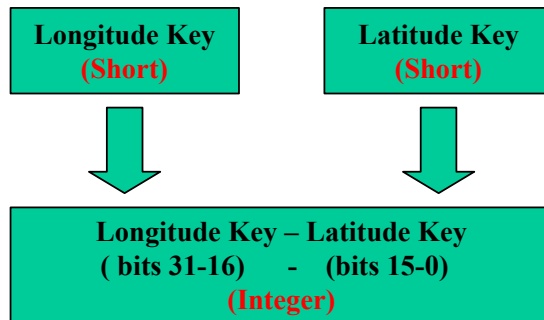


Figure 29. Mapping Two Dimensional Key into One Dimension.

- **Adjusting The Branching Factor for Each Feature**

Conversion results show that there are dramatically different distributions between feature data, even within the same coverage. For example, road data has about 700,000 primitives, railroad data has about 300,000 primitives, and miscellaneous transportation line data has about 150 primitives for the North America transportation coverage. For this reason, the B⁺-Tree structures cannot be adjusted for maximum read performance for the all features at the same time.

The solution to this problem is adjusting the branching factor of each B⁺-Tree structure for each feature according to its data. Of course, some algorithm changes in the conversion tool were required to implement this improvement, and these changes are discussed in section 4.4.

This approach allows choosing the best branching factor for each feature according to its number of primitive for better searching performance. In addition, another benefit of this approach is that it allows adjusting the depth of the B⁺-Tree for better performance when the branch factor is calculated. When the number of primitives of the feature is calculated, the branching factor can be calculated for B⁺-Tree structure with a depth of two by taking the ceiling of the square root of the number of primitives. For example, if the primitive number is 18 then branching factor becomes 4 for a tree that has a depth of two. However, a normal B⁺-Tree insert algorithm cannot be used to fill this type of tree. All the nodes must be forced to fill up, and a simple algorithm as shown in Figure 30 can fill this tree very easily because the data for the entire B⁺-Tree is available at the time of insertion.

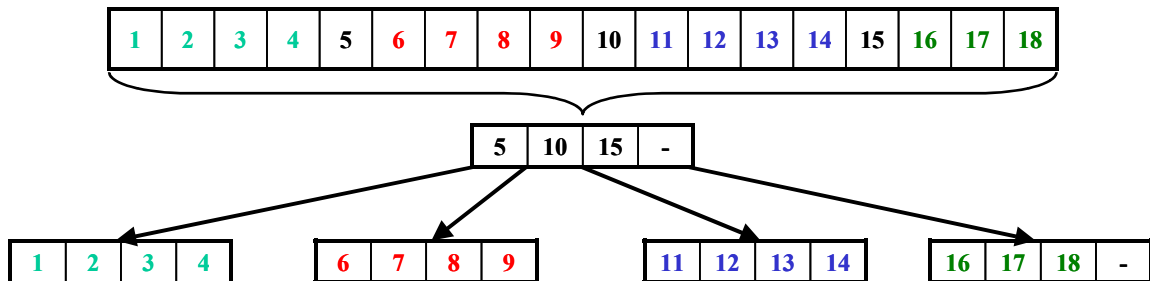


Figure 30. Forcing a B-Tree for a depth of two.

This approach allows adjusting the B^+ -Tree branching factor according to its feature's data volume for better search performance, and adjusts the depth of the B^+ -Tree for nearly constant search performance. This approach is useful to handle different features that have very dissimilar data sizes. However, the following improvement copies the cells from the neighbor B^+ -Trees to fill the empty cells, creating a new problem to calculate the branching factor. For this reason, this approach is not used in the final data model and indexing schema. In the future, the conversion tool can be improved to calculate the branching factor for the following improvement and this approach may be used again.

- **Decreasing B^+ -Tree Accesses**

Until this approach, all attempts to increase performance concentrate on decreasing the number of B^+ -Trees or reorganizing the data. However, if the number of B^+ -Tree accesses can be decreased then the overall performance can be increased significantly. For this reason, this approach tries to find ways to decrease the B^+ -Tree accesses, and it is focused to read all cells with one B^+ -Tree approach. If the cells can be linked together in a way according to their longitude or latitude cell values, then after one B^+ -Tree access all linked cells can be read without any extra B^+ -Tree accesses. The node file is already organized from west to east however there is no connection between cells which have the same longitude key value because there are different B^+ -Tree structures for each different longitude key value. This results in an extra B^+ -Tree access for each row of the viewing volume. There is a need to link the cells that have the same longitude key value vertically. However, the undetermined distribution of the geo-spatial data makes this approach almost impossible for the originally proposed indexing schema.

As shown in Figure 31, there are empty cells in the data space. Linking can be done towards the southeast but this orientation narrows the viewing area and some cells at the southwest of the viewing area (small rectangle) may not be reached. The cells may be linked towards the southwest but this orientation enlarges the viewing area and some cells at the southwest of the viewing area (small rectangle) may be reached needlessly as shown in Figure 32.

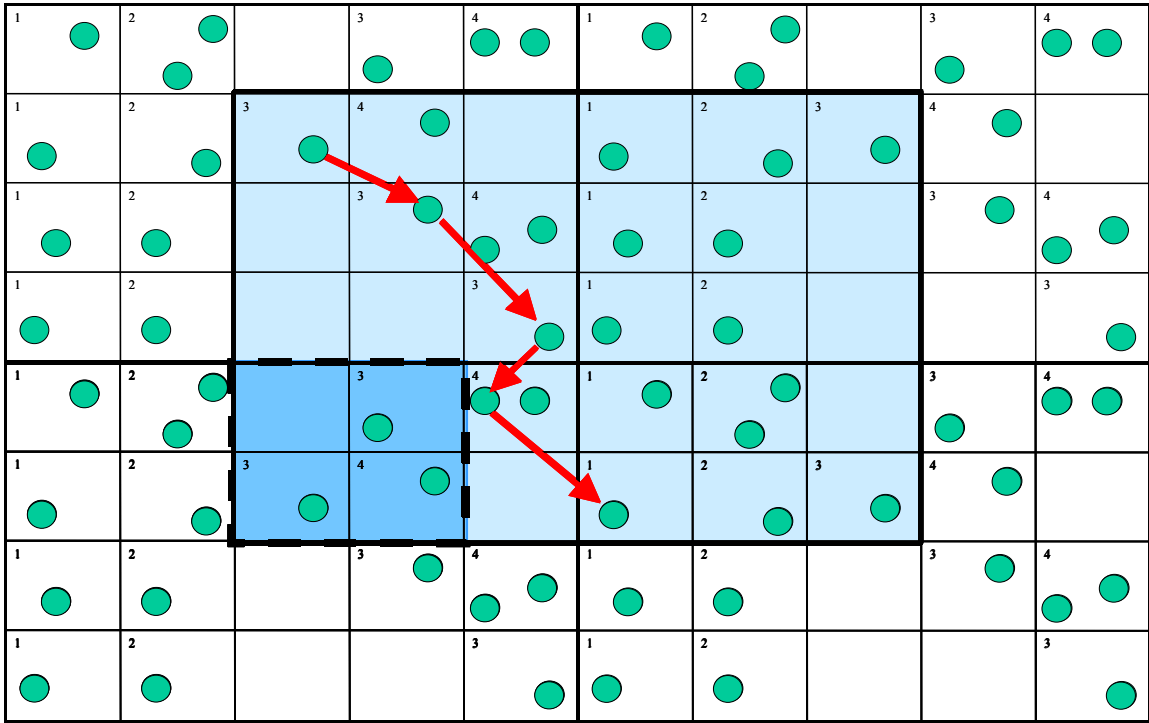


Figure 31. Linking Cells Vertically Towards Southeast.

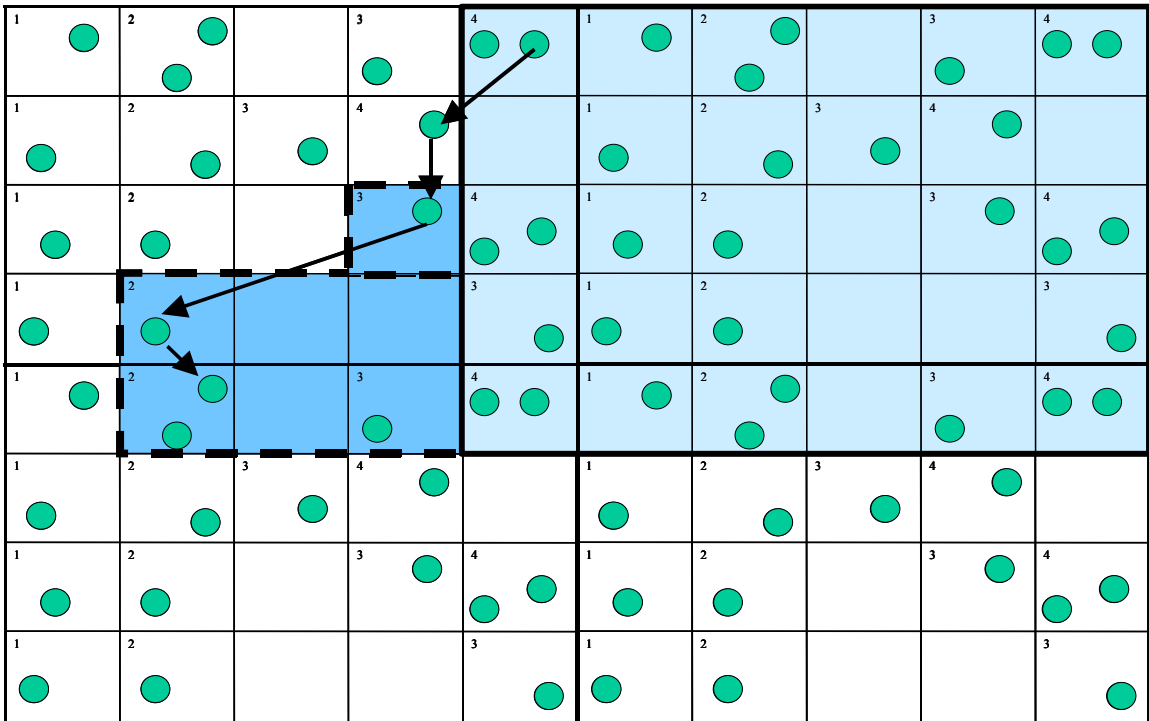


Figure 32. Linking Cells Vertically Towards Southwest.

The benefit of this approach is very significant, but the originally proposed indexing schema does not allow linking cells vertically. With some reorganization, the indexing schema may be altered for this approach. First, because the node file is organized from west to east, instead of organizing B⁺-Tree structures according to cells' latitude value they can be organized according to cells' longitude value (vertically). This means simply exchanging the indexing organization of the array structure with the B⁺-Tree structures. This new approach also connects cells vertically because all the cells with the same longitude value are contained by the same B⁺-Tree structure as shown in Figure 33.

In this new schema, because beginning cells of each row of the viewing area are contained by the same B⁺-Tree, only one access to the B⁺-Tree is sufficient for the entire viewing area. However, linked cells must also be pulled from the B⁺-Tree structure. This presents a problem in that B⁺-Trees are organized to reach a specific key quickly however when there is a need to pull out all linked cells, their performance drops and this approach offers no improvement.

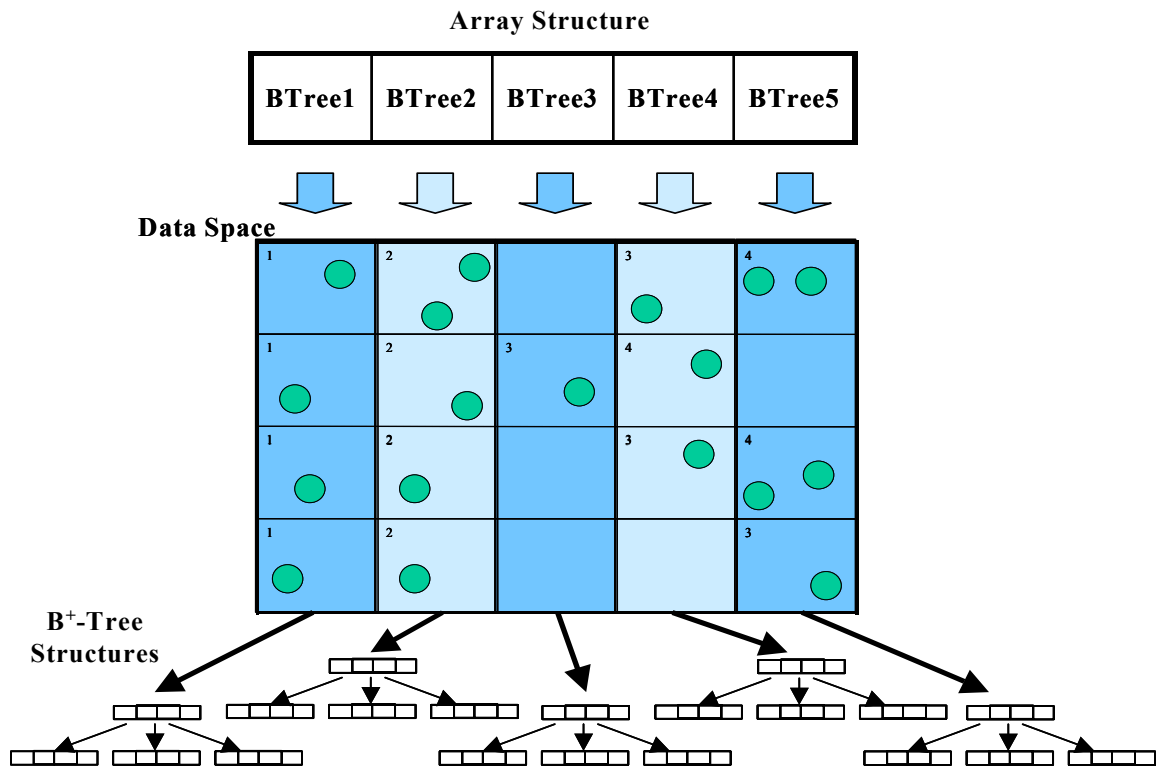


Figure 33. Final Indexing Schema.

However, the data model can be switched to the original four-file approach (including cell file). The B^+ -Trees point to a cell file where all cells are organized in an array structure. In this approach, B^+ -Tree structures can be used to find the upper left cell of the viewing area and then the cell file can be used to reach all the linked cells under the upper left cell by using the array structure of the cell file. All node data can be read from the node file, to which the cells point.

Empty cells present a problem for this approach but the solution is simple: make a copy of the right cell and use it instead of the empty cell for indexing as shown in Figure 34. This copy is a copy of the cell pointers, not the primitive data, so it only increases the size of index files, not the data file. However, this final improvement makes this approach really work.

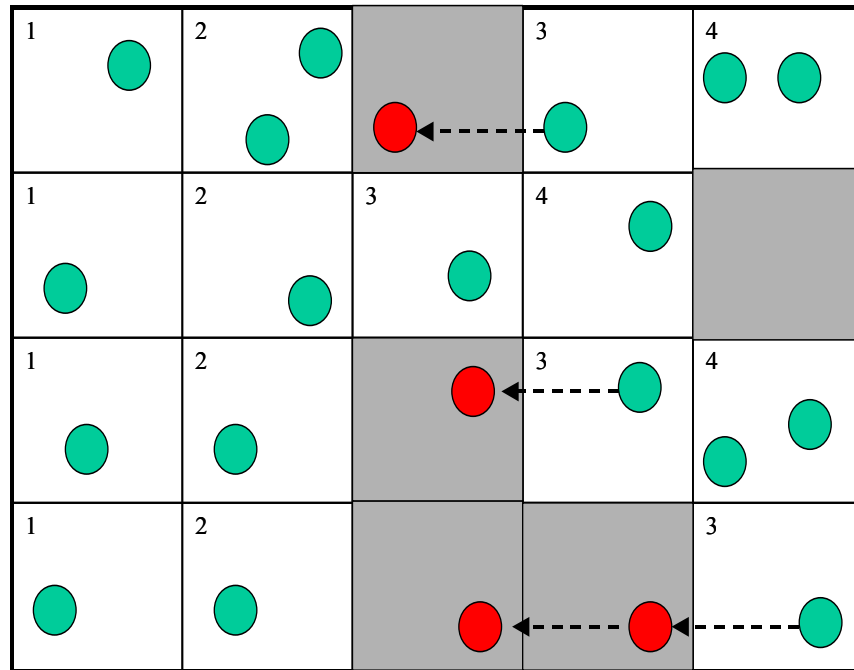


Figure 34. Solving Empty Cell Problem for the Final Indexing Structure.

Figure 35 gives an example to show how the new indexing schema works. Assume the small rectangle in the center is the viewing area. B⁺-Tree number 2 is used to reach the upper left cell, cell 3, as fast as possible. The key (cell 3) in the B⁺-Tree structure points to the location of cell 3 in the cell file. The cell file is organized like a simple array structure to read all cells under cell 3 as fast as possible. From the cell file, cells 3, 4, 5, and 6 are read into memory. Finally, cells in the cell file point to the node file for their primitives' data. This new indexing schema has larger index files than the previous ones and it uses three files for indexing (latitude index file, longitude index file, and cell file) and one file for data (node file) but its performance is much greater than the previous ones. However, it only requires minor changes over the old indexing schema when compared to performance gain obtained by this approach. As a result, this new approach requires only one B⁺-Tree access for the all scaling factors instead of having 20 to 30 B⁺-Tree accesses in the old indexing schema for 1/2,000,000 scaling and even more for high scaling factors.

After improving the indexing schema, there is a need to decrease the node file size because the original data model is twice the size of the VPF database. For this, reason the last improvement aims to decrease the size of the node file.

- **Decreasing The Node File Size**

All of the improvements listed previously aim to improve the indexing schema performance. This improvement seeks to reduce the size of the node file. Almost all of the data in the node file is necessary. The connections between the nodes are the only data that can be reduced. Because of the visibility problem, all of the connections between connected nodes are two sided. The connections between nodes are simulated in Figure 36.

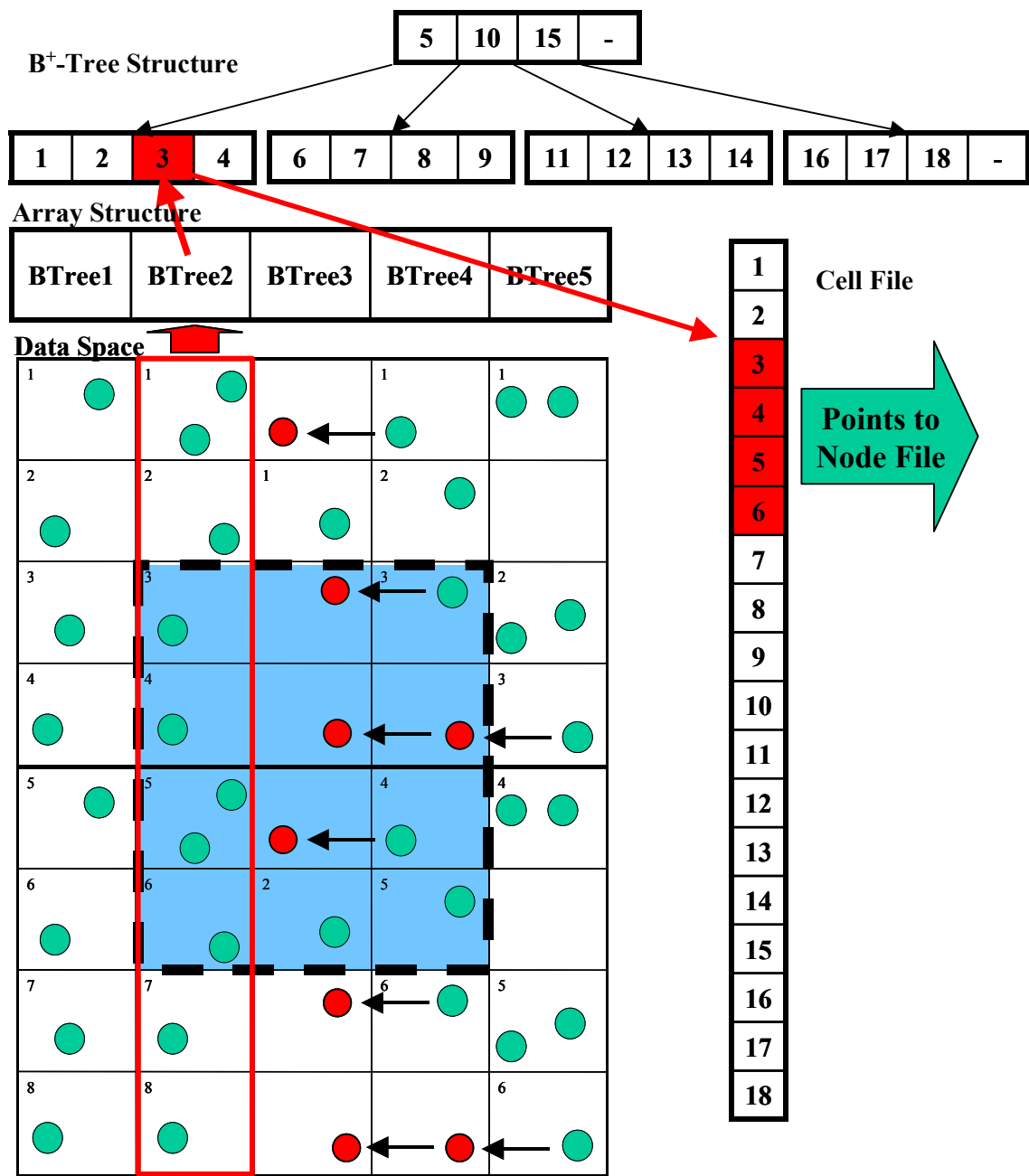


Figure 35. A Query Example.

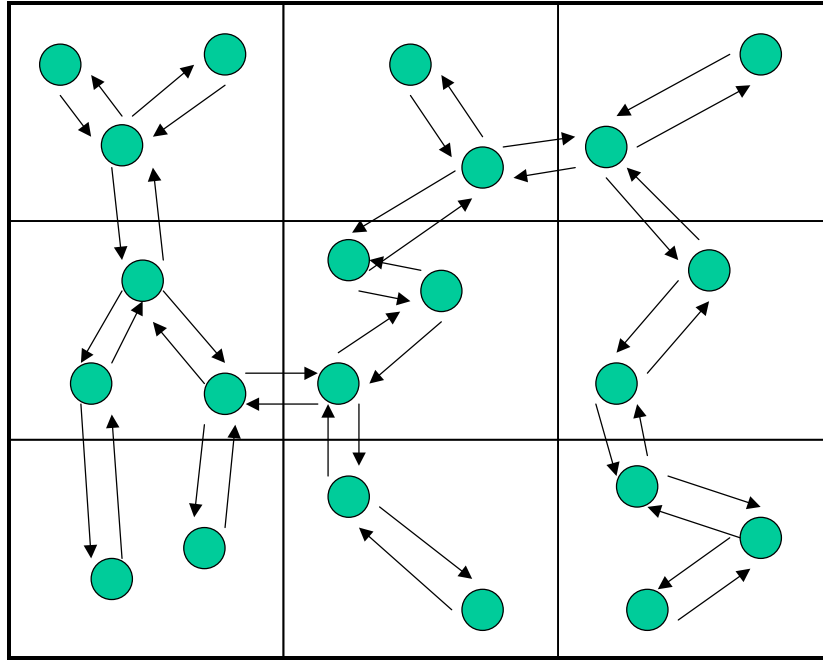


Figure 36. Representation of Connections Between Nodes.

However, all the data is read into memory cell-by-cell. This means that there is no visibility problem inside the cells because all the primitives of the cell are read and displayed at the same time. For this reason, the double links inside the cell can be broken without any loss as shown in Figure 37, and every broken link saves eight bytes from the node file. The overall decrease cannot be estimated because it changes from data set to data set. If the line segments are shorter and there are a lot of connected nodes inside a cell then the decrease may be significant. As an example, the feature “contour lines” has a node file that has a size of almost 190 Mbytes before improvement and after its size was dropped to 135 Mbytes, a 30% decrease in the data file size. This procedure is done at conversion time and requires no extra work for visualization of the data.

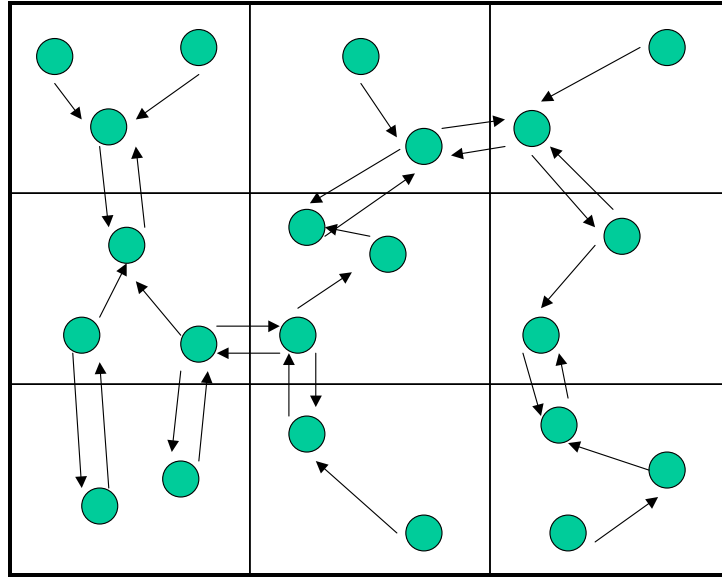


Figure 37. *Breaking One Side of the Double Links Between Nodes Inside the Cells.*

4.5 Conversion

The main purpose of conversion is to convert VPF data to the new data model and indexing schema format correctly and automatically. Conversion time is not important for the aim of this research. The conversion process takes place in two steps: preparing the old data for new model, and writing the data.

4.5.1 Data Preparation

Because the VPF data model is very different from this data model, it is necessary to prepare the data before saving it. Also, because main memory is not sufficient to hold all features, the data preparation process uses secondary storage. In this process, connections between nodes and relations between nodes and cells are organized. The new data model and indexing schema cannot be used directly for conversion in this step, because of the differences between the VPF data model and the new data model. For this reason, temporary classes are used to represent the data model at this stage. The class diagram of these temporary classes is as shown in Figure 38. As can be seen, there is no connection between temporary cell

classes and temporary node classes because all of the temporary classes are organized to work on the hard disk and not in memory.

There are three temporary files that are used to store classes for preparation: one for temporary cell data, one for temporary node data, and one for the matrix indexing structure. Their organization is shown in Figure 39. All of these files are deleted after the feature's data is converted to new data model.

This process also requires an indexing schema to reach temporary cells for a faster conversion. After trying B⁺-Tree structures for temporary indexing, a two-dimensional matrix structure was used for temporary indexing as shown in Figure 39. This matrix structure has a lot of empty cells, however this structure is just used for conversion and it is deleted afterward. The matrix structure helps to locate the cells in one disk access, however updating temporary records still requires two disk accesses. Because the VPF database is organized as a tiled hierarchy, the same cells are generally reached many times for a specific tile. For this reason, a very simple cache structure is used to decrease data preparation time.

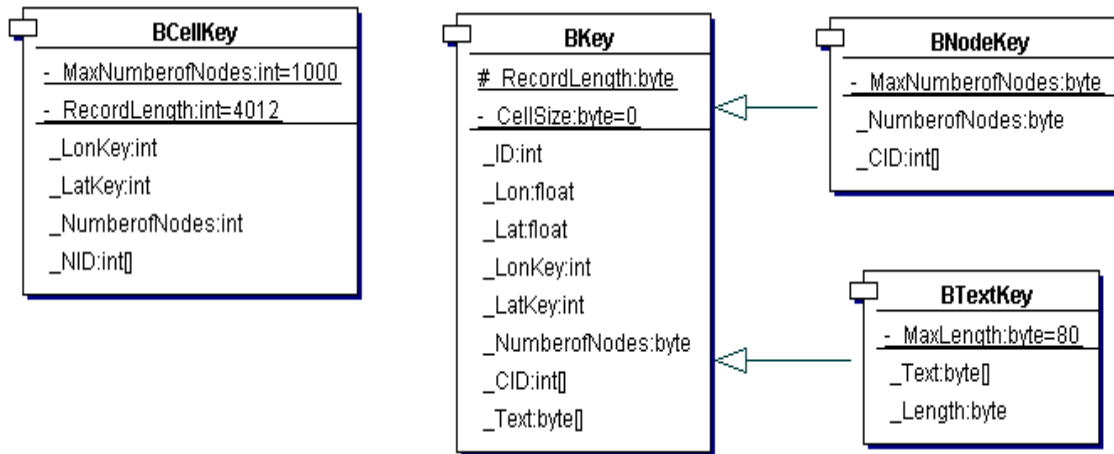


Figure 38. Class Diagram of the Temporary Classes.

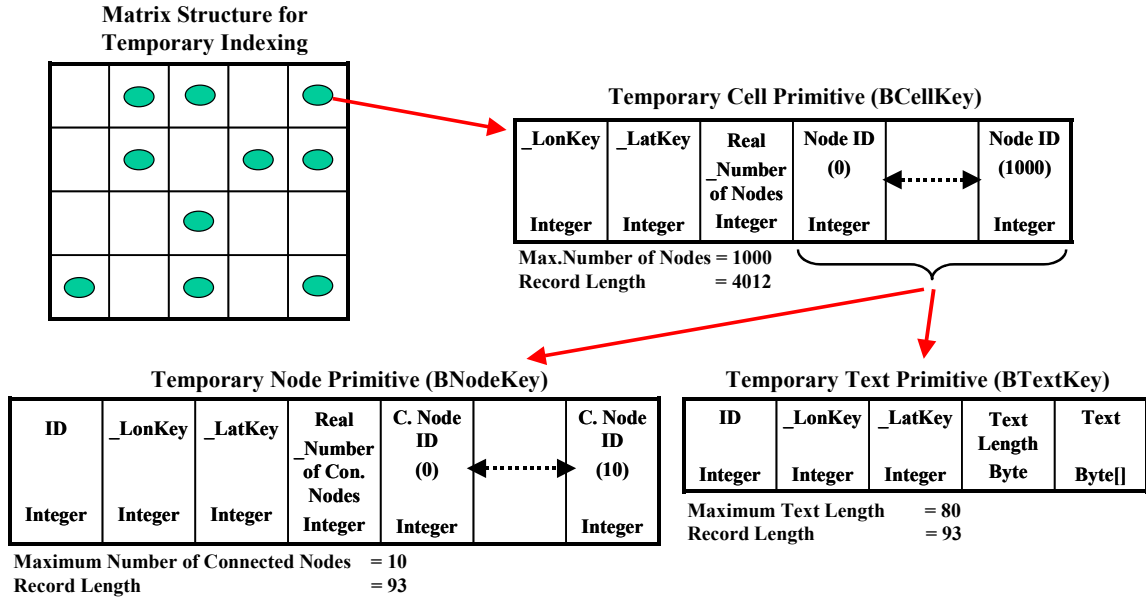


Figure 39. Temporary File Structure for Conversion.

In addition, the visibility problem is also solved in this step. Every line segment's length is calculated and compared with the predefined critical size. If the length is above the critical size then it is divided to equal length segments as described in Section 4.2.2.

4.5.2 Building The New Data Model and Indexing Schema

This step is divided into two sub-steps. The first sub-step is building the data model, and second sub-step is building the indexing schema. In the first sub-step, the data model is built from temporary files cell-by-cell using the matrix structure. The matrix structure is used to read cells from west to east. In this sub-step, the second links of connected nodes in the same cell are also broken down to reduce the data file size as described in Section 4.4. After a cell is completely built and its size is reduced, it is written to the node file and its beginning position in the node file is written back to the matrix structure for use in the second sub-step. As a result, the matrix structure is used in two main steps for two different reasons. The data model is discussed in Section 4.2 and its file structure is discussed in Section 4.3.

In the second sub-step, the array structure, the B⁺-Tree structures, and the cell file are created and built. This process is done B⁺-Tree by B⁺-Tree. For each column of the matrix structure a B⁺-Tree

structure is created in memory and filled with the related cells (another temporary cell class is used as “TempCell”) and then it is saved to latitude index file and its cells’ data is saved to a cell file as described in Section 4.4. After it is saved completely, its pointer is written to an array structure. During this process empty cells are also filled with the copy of eastern neighbor cells as described in Section 4.4. The array structure is first built in memory and when all the B⁺-Trees are created and saved to files, then it is saved to longitude index file.

4.5.3 GUI

The main purpose of the GUI is to provide a user-friendly interface and support an automatic conversion tool. This simple GUI is shown in Figure 40.

The GUI helps user to define the source path of the VPF database and destination path of the new database. The user can select desired features for conversion after the VPF database hierarchy is read. Because the whole conversion takes about 4-5 days for a VPF library, there is no button to convert the entire library once. However, the user can select more than one feature and convert them at the same time. The user must create a new file hierarchy for the new database at least once per library.

The VPF database has four libraries that cover the entire world. The user can convert all four libraries to the same directory. The tool updates the library attribute table for the database and creates the coverage attribute table for the new library. As a result, all four libraries can be stored under one database and can be used at the same time.

This tool has three classes to organize conversion in addition to all the data model and indexing classes. A singleton class “ConvertVPFData” is a unique class that organizes all conversion and interfaces between the file hierarchy and the GUI. The second class, “ConvertGUI,” organizes all GUI functions and objects. The third class is a thread class called “Runner,” that calls the required function of the “ConvertVPFData” while allowing GUI objects to respond to GUI events.

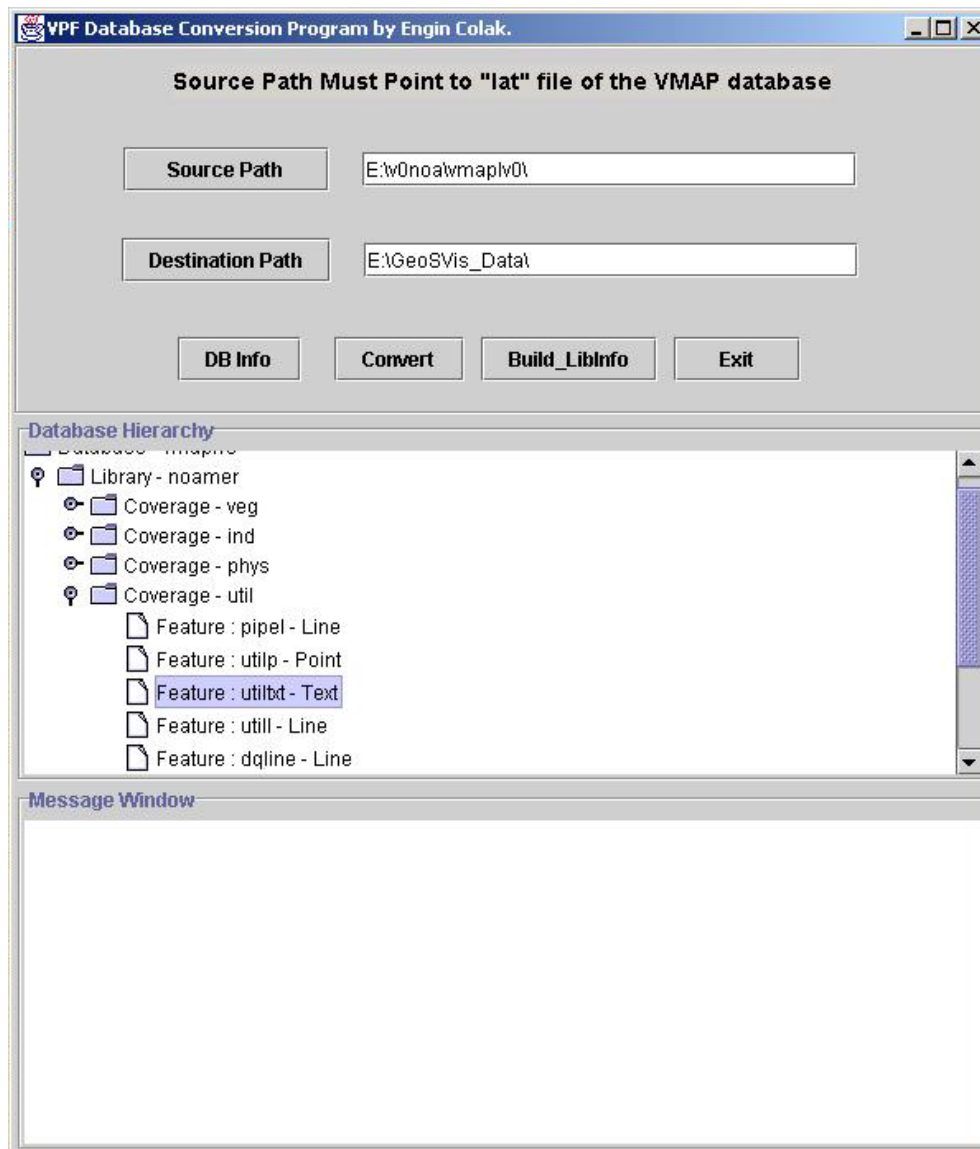


Figure 40. Conversion Tool.

The final class diagram of the conversion tool, including all data model and indexing schema classes, is shown in Figure 41.

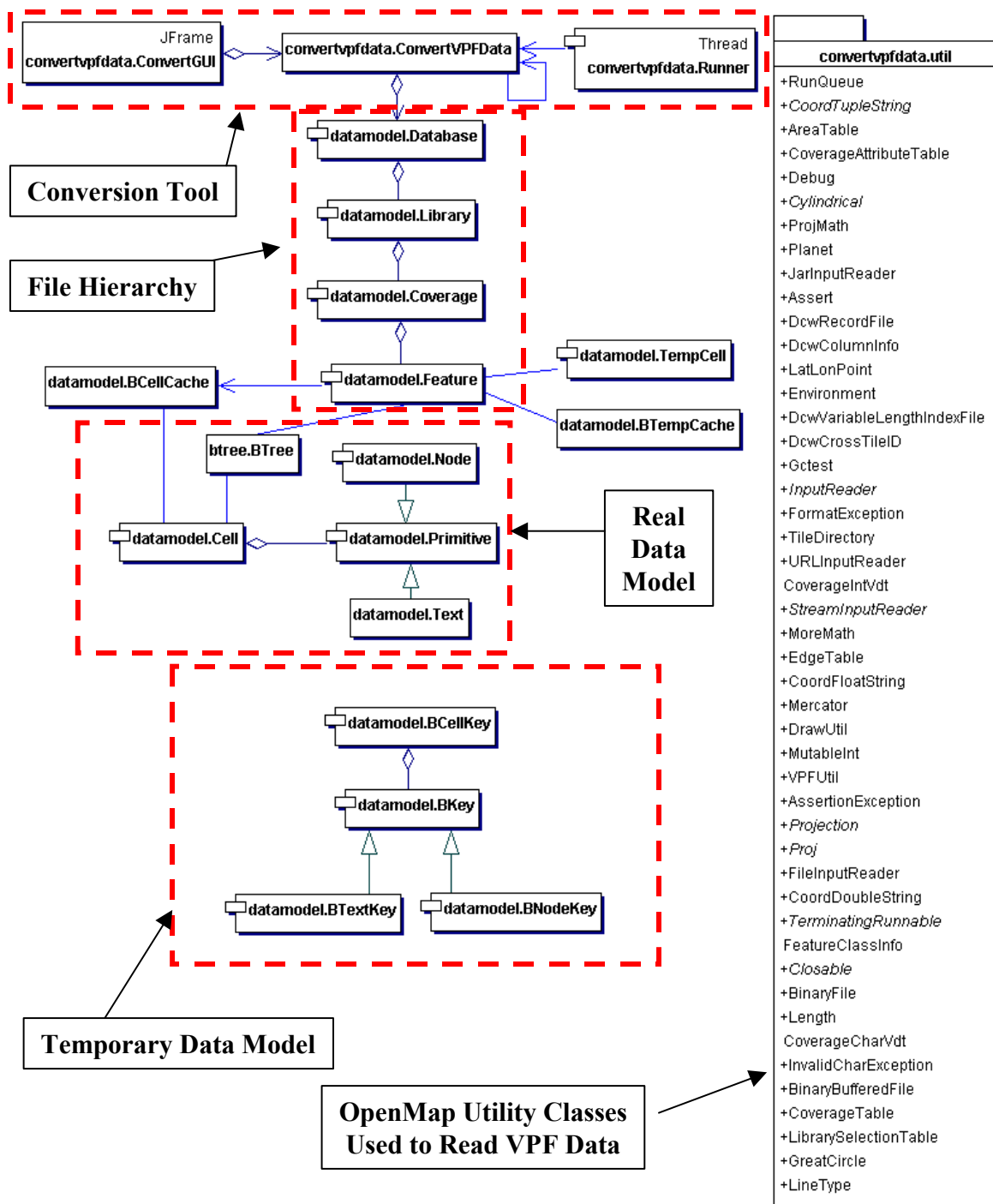


Figure 41. Final Class Diagram of the Conversion Tool.

4.6 Visualization

The visualization tool is implemented completely under the OpenMap structure. A new OpenMap layer was implemented to read and convert the new data to OpenMap graphics classes, and visualize them via projection classes. Three new classes were implemented over the new data model and indexing structure. The first class is the layer class “GeoSVis” which organizes the interface between the OpenMap GUI and the database. The second class “GeoSVisGraphicWareHouse,” creates OpenMap graphics such as points, lines, and text. The third class, “CellCache,” is a simple cache for indexing the structure to improve performance. Also some OpenMap input/output and utility classes were extended for use with the new data model.

4.6.1 GeoSVis Layer

The main purpose of the layer class in the OpenMap architecture is to prepare the database, read data from the database, and render the data. The only way to create an instance of a layer to display is to create it via a properties file. After creation, the layer can read its own properties and prepare the database for query.

Figure 42 shows how a layer can be created via a properties file. First, it must be added to the layers list by using the `openmap.layers` property. A layer also can be added to the startup layers list via the `openmap.startUpLayers` property, but all the layers in the startup layers list must also be in the layers list. All the layers in the startup layers list are displayed immediately after program startup. However, if a layer is added to the layers list but not to the startup layers list, it must be activated from the “Layers” menu to display its data. Also the layer properties must be set to create it correctly. The `LAYER.class` property defines which OpenMap class will be instantiated. The `LAYER.prettyName` property gives the layer a name to separate it from other layers. The `LAYER.geosvisPath` property defines the database path for the GeoSVis layer. The `LAYER.featureTypes` property defines which layers to display from the database.

The GeoSVis class reads the library and coverage attribute tables and builds the database file hierarchy. It prepares selected features for querying based on their defined properties. The data model and

indexing schema are the same as the conversion tool's, however, their operations are organized for reading and querying data, not for writing. If a feature is selected via the properties file then its array index structure is read into memory and its query flag is set to true.

When a layer is created, it is automatically added to the OpenMap projection listener. Any projection or view changes are directly passed to the layer. When a layer gets the projection change event, it must query the database according to the new viewing area and read the data for that area. It sends query commands to the database and the database sends them to features using the file hierarchy. Features read data using the indexing structure and send it to the "GeoSVisGraphicWareHouse" to create graphical objects. Finally, the GeoSVis layer sends the graphical objects to the current projection class to project them.

Another job of the GeoSVis layer is to prepare the GUI to set the graphical attributes of features such as color and shape. In addition, a feature can be selected or deselected for querying using the GUI.

```
# To create a layer it must be added here  
openmap.layers=daynight graticule geosvis scaledPolitical scaledFillPolitical  
  
# To display a layer at startup it must be added here  
openmap.startUpLayers=geosvis graticule scaledPolitical scaledFillPolitical  
  
## Properties of the layer must be specified  
## GeoSVis Layer  
geosvis.class=com.bbn.openmap.layer.geosvis.GeoSVis  
geosvis.prettyName=GeoSVis Contour Lines  
geosvis.geosvisPath=E:\\realdata_newapproach_4files_cell10  
geosvis.featureTypes=noamer.trans.roadl noamer.trans.railrdl  
geosvis.noamer.trans.roadl.lineColor=ffff0000
```

Figure 42. Example Properties Definitions for the GeoSVis Layer.

OpenMap uses a buffered file reader, “BufferedBinaryFile,” to read random access binary files. The GeoSVis layer extends this class uses it for its own files. Most of the general layer code is written in similar manner to that of the “VPF Layer” because these two layers are compared for the performance evaluation of the new database. The GeoSVis layer uses the same utility classes and the same design patterns for fair comparison. A brief class diagram of the GeoSVis layer is shown in Figure 43.

4.6.2 Layer Graphic Warehouse

The main purpose of the “GeoSVisGraphicWarehouse” class is to convert geo-spatial data to graphical objects (lines, points, and text). OpenMap has many graphical objects in the OM Graphics package to display the features according to their coordinate values. These objects can be directly created using coordinate values of the features instead of X-Y screen coordinates, and they perform the conversion themselves during the projection process. In addition, the OMGraphicList collection class can be used to collect all of these graphical objects in one list to organize them easily. Because features read data cell-by-cell, features can send an entire cell to the graphic warehouse at one time. The Graphic warehouse creates OM Graphics objects according to feature type and adds these objects to the OMGraphicList object. After all the data is read, this list is returned to the GeoSVis layer to be sent to the current projection class.

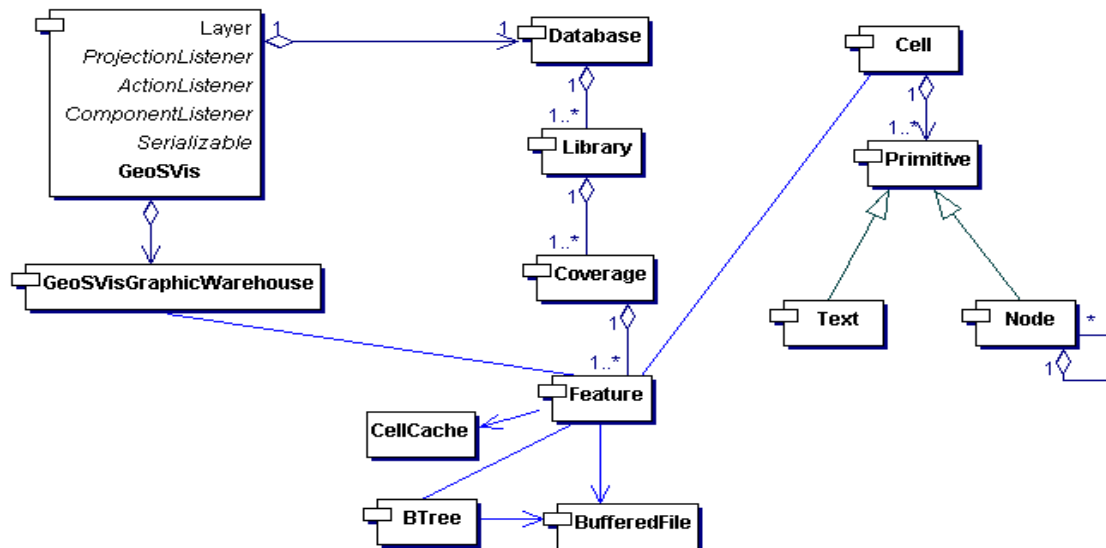


Figure 43. GeoSVis Layer Class Diagram.

4.6.3 GUI

The visualization tool has the same GUI as the OpenMap application, as shown in Figure 44.

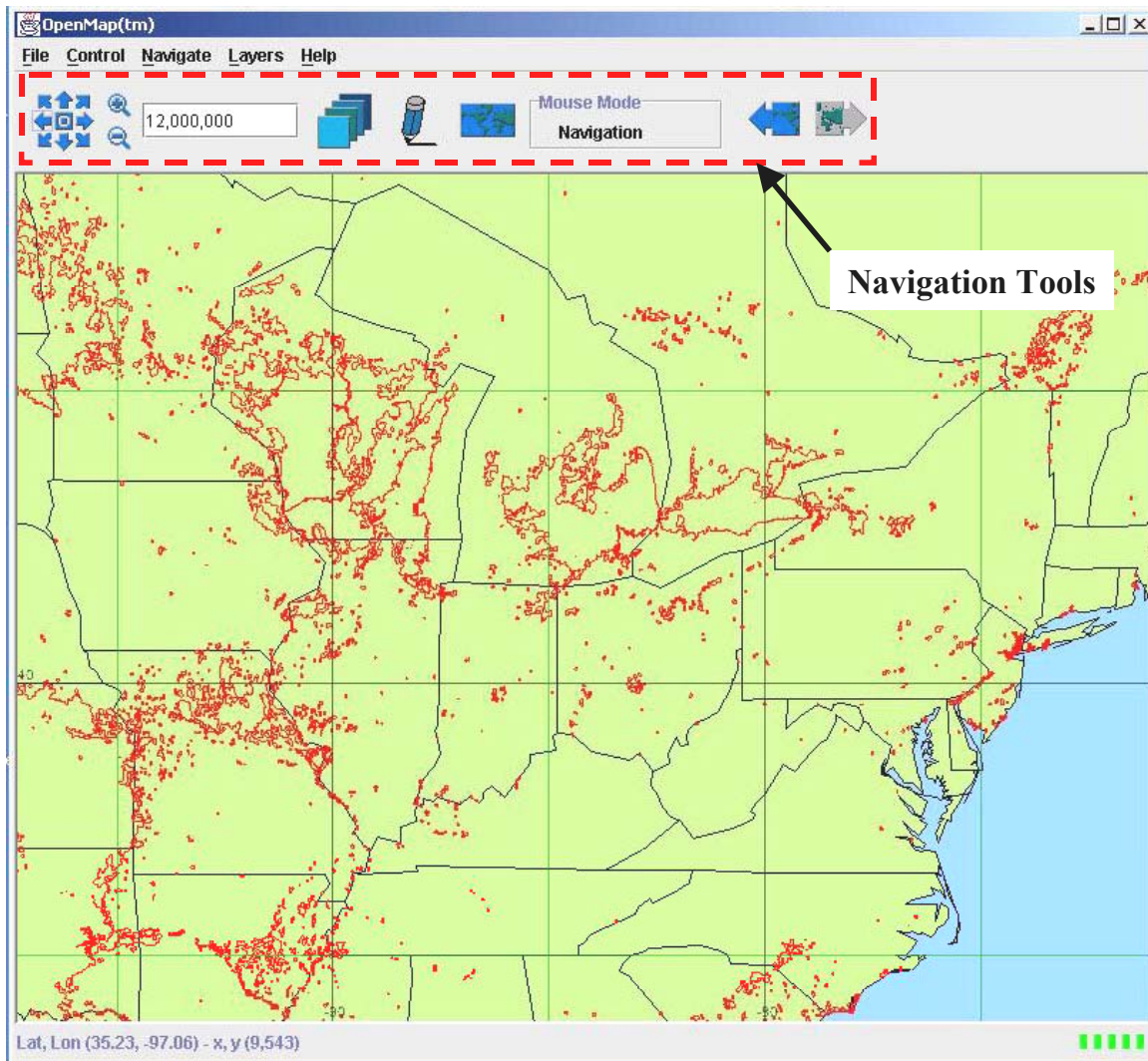


Figure 44. OpenMap GUI.

The GeoSVis layer responds to all the navigation tools seen in Figure 44. In addition, the user can add or delete features to or from the query list by using the Layers menu shown in Figure 45. A complete layer can be selected or deselected in response to projection changes from the Layers menu.

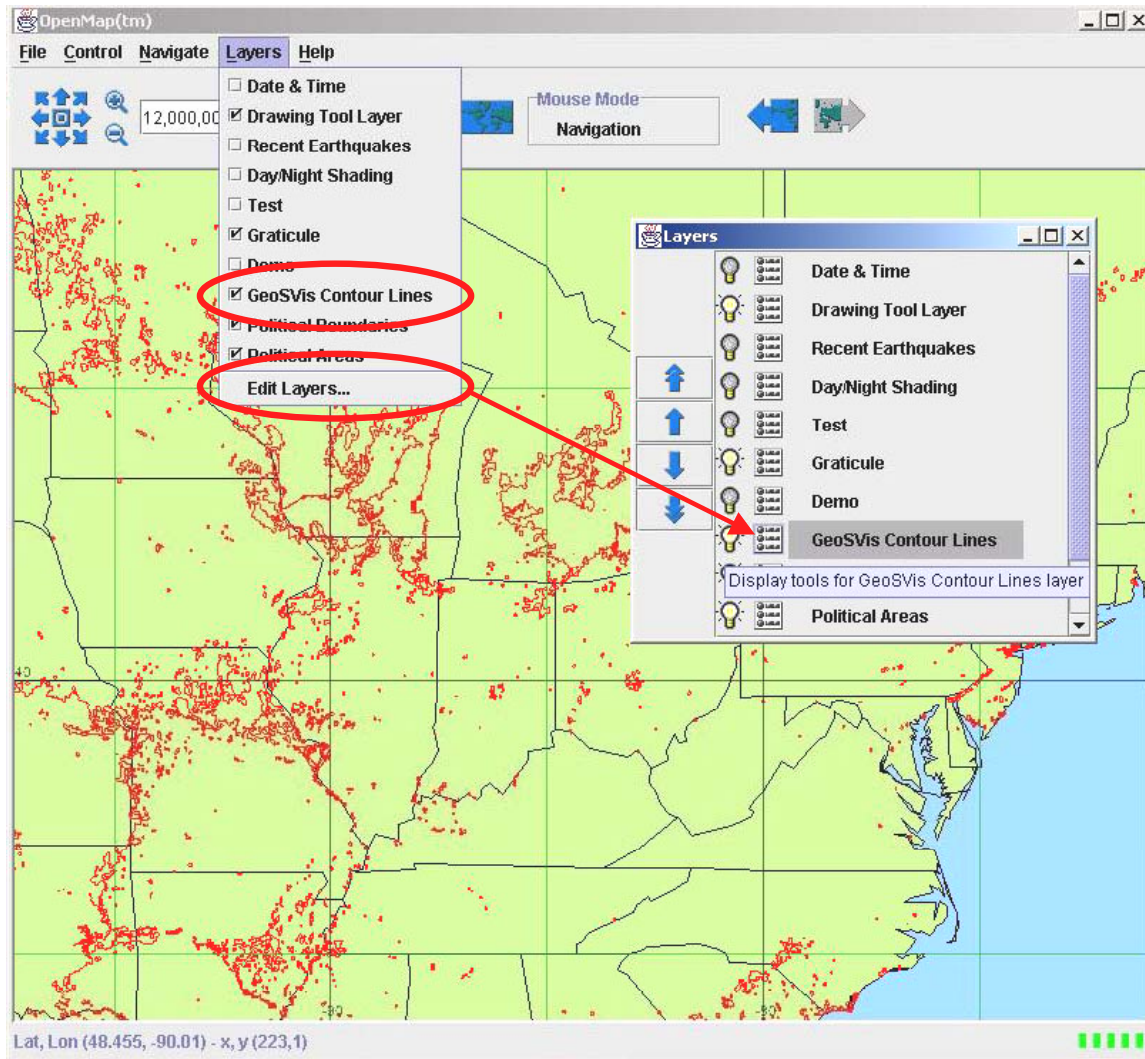


Figure 45. Changing Drawing Attributes of a Feature.

In addition, drawing attributes and the query flag of the features can be changed from the Layers/EditLayers menu item. When this menu item is selected, OpenMap passes this event to the GeoSVis layer. The GeoSVis layer is responsible for displaying the required GUI to change the drawing attributes and the query flags of the GeoSVis features shown in Figure 46.

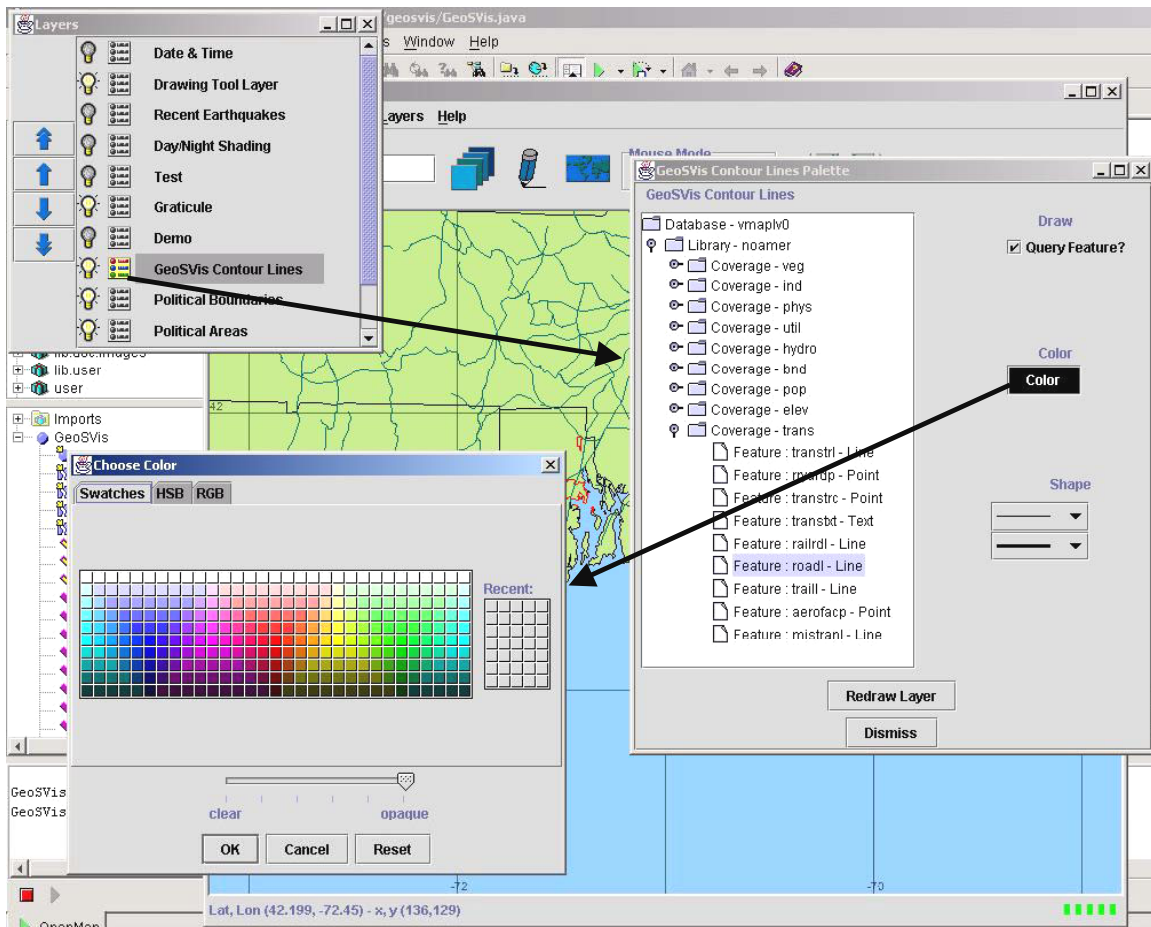


Figure 46. The GeoSVis drawing attributes GUI.

4.7 Summary

Because this research focuses on a specialized geo-spatial database for visualization applications, first its differences and characteristics according to general geo-spatial databases and applications are examined (Sections 4.1, 4.2). A new data model and indexing schema is designed for visualization applications based on this examination (Section 4.2). Then, the implementation focuses on two parts: creating the new database and required tools (Sections 4.5, 4.6), improving the new data model and indexing schema (Section 4.4).

In the first part, two tools are implemented: one tool for creating and populating the new database and one tool for visualization of the new database. The first tool, conversion tool, loads the new database

with the VPF data and builds the new data model and indexing schema during conversion. This tool and problems encountered during implementation are discussed in Section 4.5. The visualization tool reads and displays the data from the new database. This tool is implemented within the OpenMap structure as a new layer and is discussed in Section 4.6. The visualization tool is also used for the evaluation of the research as discussed in the next chapter.

After the implementation of the first part, the second part is intensified to improve the new data model and indexing schema (Section 4.4). The second tool is used to compare the performance improvement of the new database with the VPF database in this part. Improvements are focused on decreasing the B⁺-Tree accesses and the number of B⁺-Trees required querying the new data for a map view for a better indexing schema. In addition, the new data model is also examined to decrease the size of the data file for faster reads. The aim of this part was to improve the new data model and indexing schema to reach the target data quickly and to read the bunch of data in shortest time. The improvements that are made in this part form the current data model and indexing schema.

V. Results

5.1 Introduction

Implementation of the new layer “GeoSVis” gives OpenMap the capability to access both VPF data and the new database. As noted in Chapter 4, the layer structure and buffered files used to access VPF data are mirrored in the new layer. This allows the use of OpenMap to perform a fair comparison of the two database formats. This chapter examines and discusses the test results.

5.2 Testing Method

OpenMap is designed to read and project geo-spatial data to the user via its GUI. However, this research is focused on just the “reading” time, not the “projection” time. It is very difficult to obtain fair tests using the OpenMap GUI due to its event-driven nature. As a result, three additional functions for each database were implemented for testing purposes. These functions were one additional constructor, one new `getRectangle` function, and one main function for each database. These functions were identical for both databases. The new constructor allows creating databases without reading the properties file. The new `getRectangle` function allows querying both databases without projection. Finally, the main function organizes test patterns and writes statistical results to files.

Testing was performed in five different patterns. The goal of these patterns was to simulate user interaction with the GUI.

The results were gathered separately for each pattern and are discussed in the following subsections. Each pattern was repeated one hundred times to make the results statistically correct, and their average is used for evaluation. In addition, the features were divided into several categories according to their size since there are many features in the database and every feature has a different data size. This categorization, shown in Table 4, helps in understanding the results. Unfortunately, the OpenMap VPF layer does not support point features. For this reason, an evaluation of them could not be done. However,

their data sizes are very small when compared with the other features and their characteristics are similar to those of text features. As a result, the performance of text features approximately reflects the performance of point features.

5.3 Test Patterns

Five different patterns were used to simulate different characteristics of user interaction with the map visualization tools. These patterns and their results are discussed in this section. Each test result shows the average speed-up of the new database for the related pattern when compared to the VPF database as shown in Equation 1. The test results (Tables 5, 6, 7, 8, and 9) for each pattern are organized by the feature categories listed in Table 4 and show the speedup of the new data model based on Equation 1.

Table 4. Categorization of Features for Testing.

Categories	Number Of Primitives (x1000)	Text Feature Type	Line Feature Type	Area Feature Type
1	Under 2.5	bnd.bndtxt	trans.mistranl	pop.mispopa
		hydro.hydrotxt	util.dqline	
		ind.indtxt		
		phys.phystxt		
		pop.poptxt		
		trans.transtct		
2	2.5-50		hydrp.aquecanl	ind.extracta
			trans.transrsl	
			util.pipel	
3	50-150		bnd.polbndl	veg.swampa
			trans.trail	phys.grounda
				pop.builtupa
4	150-250		bnd.depthl	phys.landicea
			util.utill	phys.seaicea
5	250-1000		bnd.coastl	bnd.oceansea
			trans.railrdl	bnd.polbnda
			trans.roadl	veg.cropa
				veg.grassa
				veg.tundraa
6	1000-2000		hydro.watcrsl	veg.treesa
7	Over 2000		elev.contourl	hydro.inwatera

$$SpeedUp = \frac{AveragePatternTimeforVPFDatabase}{AveragePatternTimefortheNewDatabase} \quad (1)$$

5.3.1 Pattern One

This pattern simulates a user who searches the map at the same zoom level, and shifts their view half the window size in any direction. It is assumed the user would continue to shift (or “pan”) their view as they explore the map. The search pattern is described in Figure 47, and there are 141 shifts for the entire pattern. The standard zoom level (scaling) is 1 / 2,000,000 for this pattern.

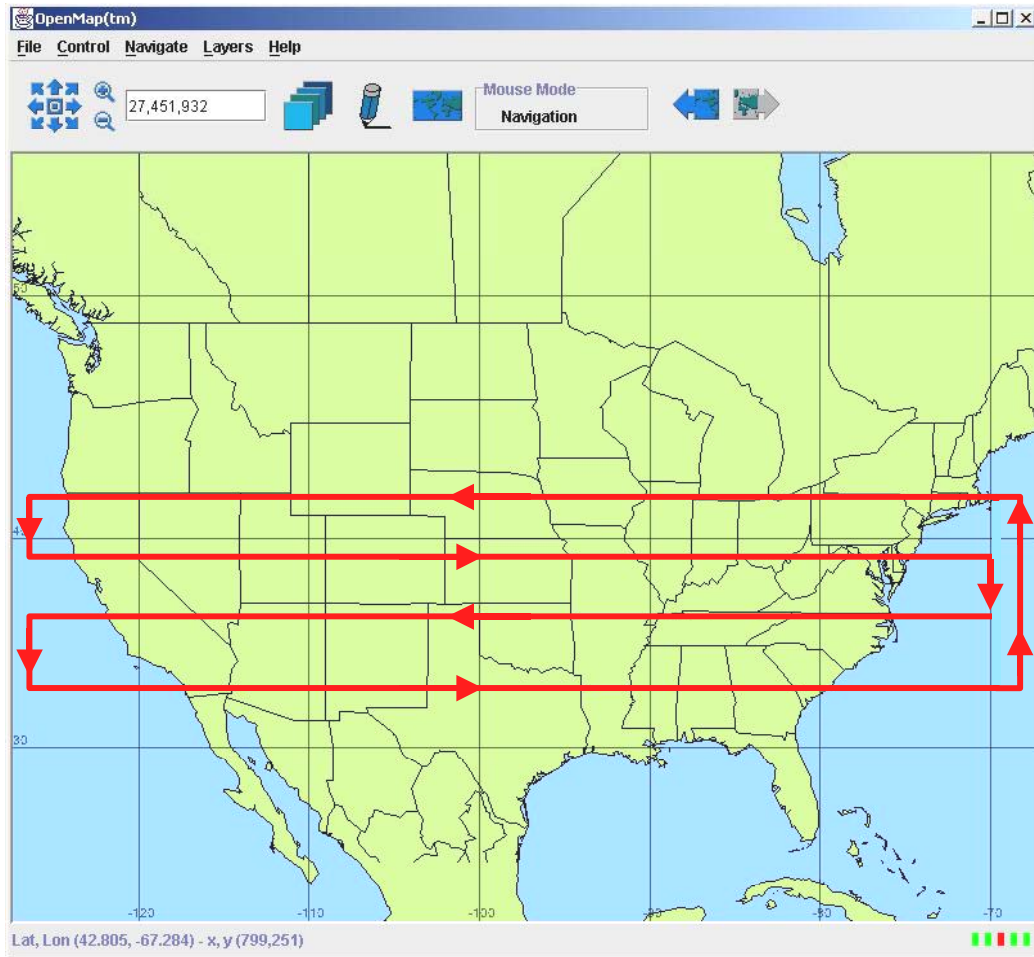


Figure 47. Test Pattern One.

Test results from Table 5 points out two important cases: the performance gain changes according to the feature type and the data size. The performance gain of “text” features are less than the other features. Because their data size is small, the new indexing schema does not have the opportunity to offer any performance gain. Also, “Text” features in the VPF database are kept in different files from the other features, and this organization makes it possible for VPF-based access to reach the data faster. The VPF database can easily reach “text” features and then read all the data at once because of its small size. However, the query time for this feature type is generally very fast (under 0.002 seconds) for both databases.

There are two reasons for the performance difference between “line” and “area” features. The first reason is once again related to how VPF stores its information. The VPF database stores all the data associated with a feature type in the same file. Because the data size is quite different between the “line” and “area” features (2500 to 2,000,000 primitives), the VPF database spends much of its reading time separating the features from each other. This time loss becomes very significant for the smaller features. For this reason, the performance gain from these small features is not included to calculate the second overall result for all patterns.

Table 5. Test Results for Pattern One.

Improvement by Feature

Categories	Number of Features	Feature		
		Text	Line	Area
Under 2500	6/2/1	8.67	1999.93	3624.47
2500 - 50000	0/3/1	-	349.45	7.96
50000-150000	0/2/3	-	57.69	18.77
150000-250000	0/2/2	-	8.25	4.51
250000-1000000	0/3/5	-	18.54	15.76
1000000-2000000	0/1/1	-	7.90	23.15
Over 2000000	0/1/1	-	3.36	12.80
Overall Result 1 (1)	34	-	349.30	529.63
Overall Result 2 (2)	28	8.67	17.57	13.83

(1) Overall result 1 is the average of the all categories.

(2) Overall result 2 is the average of the last five categories for line features and last six categories for area features.

The second reason the performance changes is the data size itself. The new database loses its performance gain significantly when the data size increases due to the overhead of the new schema. However, its performance gain is 3.4 times faster even for the largest feature (8,000,000 primitives) in the library.

The overall test results of this pattern are 8.6 times faster for “text” features, 17.5 times faster for “line” features, and 13.8 times faster for “area” features.

5.3.2 Pattern Two

This pattern simulates a similar user action similar to Pattern One, except the user changes the view by a full window size each shift. There are 71 shifts for this pattern and the zoom level is 1 / 2,000,000. The results are as shown in Table 6.

The test results of this pattern are almost the same as Pattern One. Some minor differences are the result of the data organization of the VPF database.

Table 6. Test Results for Pattern Two.

Improvement by Feature

Categories	Number of Features	Feature		
		Text	Line	Area
Under 2500	6/2/1	9.75	2867.27	102.92
2500 - 50000	0/3/1	-	206.39	8.22
50000-150000	0/2/3	-	57.41	18.43
150000-250000	0/2/2	-	8.12	4.90
250000-1000000	0/3/5	-	17.48	15.71
1000000-2000000	0/1/1	-	7.56	22.47
Over 2000000	0/1/1	-	3.31	12.81
Overall Result 1 (1)	34	-	452.50	26.49
Overall Result 2 (2)	28	9.75	17.26	13.76

(1) Overall result 1 is the average of the all categories.

(2) Overall result 2 is the average of the last five categories for line features and last six categories for area features.

5.3.3 Pattern Three

This pattern simulates a user who searches a location using a zoom in and out pattern. The user changes the zoom level from 1 / 2,000,000 to 1 / 16,000,000 and back to 1 / 2,000,000 while searching. After the zoom level reaches 1 / 2,000,000 again the user shifts the view by half a window. This is repeated for 10 iterations. There are 132 shifts for this pattern, and the test results are shown in Table 7.

The main purpose of this pattern is to understand the reaction of the new database to the zoom changes. The data size of the features affect the performance of this pattern significantly when compared to the previous patterns. The performance gain is increased significantly for small features and decreased significantly for large features. This indicates that while the new database reaches the target data faster, it spends more time reading than the VPF database. In order to properly examine this performance drop, Pattern Four and Pattern Five expand the zoom level and then repeat the window shift route of Pattern One.

Table 7. Test Results for Pattern Three.

Improvement by Feature

Categories	Number of Features	Feature		
		Text	Line	Area
Under 2500	6/2/1	38.65	41859.25	1926.87
2500 - 50000	0/3/1	-	17291.96	51.04
50000-150000	0/2/3	-	82.35	95.94
150000-250000	0/2/2	-	5.11	2.54
250000-1000000	0/3/5	-	4.24	33.64
1000000-2000000	0/1/1	-	4.12	4.86
Over 2000000	0/1/1	-	0.88	4.46
Overall Result 1 (1)	34	-	8463.99	302.76
Overall Result 2 (2)	28	38.65	18.52	32.08

(1) Overall result 1 is the average of the all categories.

(2) Overall result 2 is the average of the last five categories for line features and last six categories for area features.

5.3.4 Pattern Four

This pattern is identical to Pattern One, with the zoom level set to 1 / 4,000,000. There are 138 shifts for the entire pattern and the test results are shown in Table 8.

The test results show the performance drop clearly when compared to the Pattern One. As in Pattern Three, the performance gain is increased significantly for small features and decreased significantly for large features. Although the overall average performance gain is almost nine times for all features, the gain drops for large data sets.

5.3.5 Pattern Five

This pattern is identical to Pattern One, with the zoom level set to 1 / 8,000,000. There are 138 shifts for the entire pattern and the test results are shown in Table 9.

Table 8. Test Results for Pattern Four.

Improvement by Feature

Categories	Number of Features	Feature		
		Text	Line	Area
Under 2500	6/2/1	9.62	86195.40	62.35
2500 - 50000	0/3/1	-	213.88	7.65
50000-150000	0/2/3	-	35.35	25.46
150000-250000	0/2/2	-	3.87	3.18
250000-1000000	0/3/5	-	7.16	7.80
1000000-2000000	0/1/1	-	3.04	9.20
Over 2000000	0/1/1	-	1.28	5.71
Overall Result 1 (1)	34	-	12351.42	17.34
Overall Result 2 (2)	28	9.62	9.53	9.83

(1) Overall result 1 is the average of the all categories.

(2) Overall result 2 is the average of the last five categories for line features and last six categories for area features.

Table 9. Test Results for Pattern Five.

Improvement by Feature

Categories	Number of Features	Feature		
		Text	Line	Area
Under 2500	6/2/1	152.15	256807.92	24.43
2500 - 50000	0/3/1	-	130.56	2.97
50000-150000	0/2/3	-	15.00	17.82
150000-250000	0/2/2	-	1.43	0.56
250000-1000000	0/3/5	-	2.35	8.29
1000000-2000000	0/1/1	-	1.72	3.98
Over 2000000	0/1/1	-	0.45	3.40
Overall Result 1 (1)	34	-	36708.49	8.78
Overall Result 2 (2)	28	152.15	3.85	6.17

(1) Overall result 1 is the average of the all categories.

(2) Overall result 2 is the average of the last five categories for line features and last six categories for area features.

The test results for this pattern show the same characteristics as the previous test pattern. However, performance change versus the data size becomes clearer. This test pattern shows that the new data model loses its performance gain for large data sets at zoom levels 1 / 8,000,000 and higher. However, it is still better than VPF for medium and small data sets. The increase in the performance of small features at high zoom levels describes a characteristic of the new database. Its indexing schema has the ability to reach and read small target data very quickly, however its data model and indexing schema loses this ability when reading large data sets.

5.4 Conclusions

After test patterns one to three were done, two characteristics of the new data model were observed. First, performance drops at high zoom levels. Second, the data size of the features affects the database's performance. For these reasons, test patterns four and five were added to evaluate the database's performance at high zoom levels. Large performance losses were found at high zoom levels, especially higher than 1 / 8,000,000. In addition, the data size of the features affects performance at high zoom levels more than it does low zoom levels. However, when the overall performance of the features are evaluated for Pattern Five, line features are 3.8 times faster than VPF, and area features 6.2 times faster than VPF.

The performance loss for high zoom levels is considered acceptable. When the zoom level becomes higher, it is necessary to read more data in short time, not to reach the data in short time. This tends to work against the new indexing structure because the overhead of the indexing structure becomes a significant cost factor.

The contour lines feature is the largest in the database (see Table 10) -- over four times larger than the next biggest feature. In addition, it is the only line feature in the elevation coverage. This means that the VPF database does not lose any time separating it from the other line features. Also, most of the data that is read by the VPF layer from the same tile is used saving even more time. Because of its large data size and organization in the VPF database, this makes the contour line feature unique, and therefore its not unexpected that performance gains are lower.

Table 10. Critical Features.

Features	Feature Type	Number of Primitives
elev.contourl	Line	8,277,266
bnd.depthl	Line	160,774
util.utill	Line	207,493
phys.landicea	Area	252,578
phys.seaicea	Area	160,747

As a result, the new data model and indexing structure loses its advantage at high zoom levels for some features, particularly those with large data sizes. Even in those conditions, however, the overall performance of the new data model and indexing schema is better than the VPF database. For example, it is two times slower than VPF database for contour lines at zoom level $1 / 8,000,000$. However, it is approximately three times faster than other line features at the same zoom level.

VI. Conclusions and Future Work

6.1 Conclusion

During this thesis, a simple data model, an indexing schema based on this simple data model, a conversion tool for new data model, and a visualization tool were produced to support visualization applications such as mission planning and combat simulation. The main goal was to build a low-cost and high-performance geo-spatial database to support these applications. A secondary goal was to build a platform-independent database that would support as many applications as possible. These two goals were achieved, and all the products of this thesis are platform independent and do not require any commercial DBMS support.

The main goal of this thesis was to improve the performance of geo-spatial databases for visualization applications as discussed in Chapter Five. This goal was reached for zoom levels less than $1 / 8,000,000$. For example, the average performance improvement for text features is 8.6, for line features 17.6, and for area features 13.8 at a zoom level of $1 / 2,000,000$. However, this performance drops at high zoom levels ($1 / 8,000,000$ and higher) for features that have large data sizes. For contour lines the average performance improvement drops from 3.4 at zoom level $1 / 2,000,000$ to 1.3 at zoom level $1 / 4,000,000$ and 0.5 at zoom level $1 / 8,000,000$. However, this performance drop is not standard for all features. For example, despite the performance drop for the contour line feature, the average performance improvement for all line features is 3.9 at zoom level $1 / 8,000,000$. On the other hand, a digital map is generally prepared by combining several features at the same time to form a meaningful view. For this reason, combining several features for one view may stabilize the performance drop. For example, if a map view contains political boundary lines, coastlines, railroad lines, road lines, contour lines, political boundary areas, and water areas then the performance improvement for this view is approximately 5 at $1 / 8,000,000$ zoom level based on the test results.

6.2 Future Work

This research generally focused on the data model and indexing schema. For this reason, beside the data model and indexing schema future work may focus on the visualization and the caching of the new data model.

6.2.1 Data Model and Indexing Schema

The data model may be reorganized for large cell sizes for better performance at high zoom levels. In addition, instead of a combination of array and B⁺-Tree structures, another two-dimensional indexing structure may be able to reach data faster and read it in shorter time.

Even though some features are small, the indexing structure occupies a large amount of space. For this reason, a new approach may be able to change the cell size for small features and therefore decrease the index file size. Small features may also be combined together to increase performance and decrease index file sizes.

6.2.2 Conversion

The conversion process takes a significant amount of time to run. Using more complex caching methods may decrease this time, and buffered files may also be used for further gains. Statistical data can be collected during the building of the node file and then the indexing structure may be created more precisely by analyzing these statistics.

6.2.3 GeoSVis Layer

Except for reading data from the new database, all visualization is performed by using the OpenMap classes. Creating more specific graphic and projection libraries for the proposed data model may help to improve both reading and visualization performance. Because the new data model is based on the primitives and these primitives are used to represent the features alone without any relations, the new data

model cannot fill the inside of the “area” features during visualization. However, new graphic libraries may solve this problem by combining the line features in order to create polygons after the data is read.

In addition, a more complex buffered file may improve reading times especially for different zoom levels. The current buffered file does not allow changing the data size after it is created, however the data size changes dramatically according to zoom level. If the data size can be adjusted before reading, then the performance for higher zoom levels may be increased.

Because of the organization of the new data model and indexing schema, they allow cell caching for both data and indexing. However, the current caching method is very primitive. Improving the cache mechanism would result in further performance gains.

6.3 Summary

Because of the characteristics of the geo-spatial data and the complex query commands of the geo-spatial databases, geo-spatial databases have a significant performance problem in general when compared to the traditional databases. This performance problem affects the all applications that use geo-spatial databases. However, there are some visualization applications that do not need the complex spatial query commands and data model, but these applications still have to pay the performance penalty of the geo-spatial databases. For this reason, a special geo-spatial database may be build to improve the performance of these visualization applications such as mission planning and combat simulations.

The main approach is to organize the geo-spatial data model as simple as possible that will be sufficient for the query needs of the visualization applications, then to build two-dimensional high performance indexing schema based on these simple data model. For this reason, instead of complex features data model is built on simple geo-spatial primitives, then a combination of an array and B⁺-Tree structures are used together to index these simple primitives. In a way, while much research on the geo-spatial databases are focused on high-dimensional indexing structures based on complex data models to improve the performance of the geo-spatial databases in general, this research is focused on two-

dimensional high-performance indexing structure based on spatial primitives to improve the performance of the geo-spatial databases for visualization.

This approach based on simple data model and two-dimensional high-performance indexing structures gave very good results especially for low zoom levels. Performance improvement is about 10 times the VPF database for all features in zoom level 1 / 2,000,000. Although, the performance improvement drops for large data sets while the zoom level increases, in general this approach has a promising future.

Bibliography

- (BBNT, 2001) BBN Technologies Solutions. Open Systems Mapping Technology (OpenMap): <http://openmap.bbn.com>, 2001
- (Berchtold, 2000) Berchtold, Stefan, Daniel A. Keim, Hans-Peter Kriegel, and Thomas Seidi. "Indexing the Solution Space: A New Technique for Nearest Neighbor Search in High-Dimensional Space," *Knowledge and Data Engineering, IEEE Transactions on*, 12/1: 45-57 (January/February, 2000).
- (Bertino, 1999) Bertino, Elisa and Beng Chin Ooi. "The Indispensability of Dispensable Indexes," *Knowledge and Data Engineering, IEEE Transactions on*, 11/1: 17-27 (January/February, 1999).
- (Brinkhoff, 1993) Brinkhoff, T., Hans-Peter Kriegel, and B. Seeger, "Efficient Processing of Spatial Joins Using R-trees," *SIGMOD Conf., Proceedings of ACM*, 1993.
- (Camara, 1999) Camara, Gilberto, Rogerio Thome, Ubirajara Freitas, Antonio Miguel, and Vieira Monteiro. "Interoperability in Practice: Problems in Semantic Conversion from Current Technology to OpenGIS," *Interoperating Graphic Information Systems Second International Conference, INTEROP' 99*: 129-138, 1999.
- (Cattenstart, 1999) Cattenstart, Frits C. and Henk J. Scholten. "Towards OpenGIS Systems", *Interoperating Graphic Information Systems Second International Conference, INTEROP' 99*: 163-176, 1999.
- (Chen, 1999) Chen, Chung-Min and Rakesh K. Sinha. "Raster-Spatial Data Declustering Revisited: an Interactive Navigation", *Data Engineering, Proceedings. 15th International Conference on*: 600-607, 1999.
- (Cicerone, 2000) Cicerone, Serafino, Daniele Frigioni, Laura Tarantino, and Paolino Di Felice. "Interacting with Topological Invariants of Spatial Databases," *Database Applications in Non-Traditional Environments (DANTE'99), Proceedings International Symposium on*: 213-217, 2000.
- (Cormen, 2000) Cormen, Thomas H., Charles E. Leiserson, and Ronald L. Rivest – Introduction to Algorithms. London: The MIT Press, 2000.
- (Cranston, 1999) Cranston, Charles B., Frantisek Brabec, Gisli R. Hjaltason, Douglas Nebert, and Hanan Samet. "Adding an Interoperable Server Interface to a Spatial Database: Implementation Experiences with OpenMap," *Interoperating Graphic Information Systems Second International Conference, INTEROP' 99*: 115-128, 1999.
- (DoD, 1996) Department of Defense. Interface Standard for Vector Product Format. MIL-STD-2407: 1996.

- (Doyle, 1999) Doyle, Allan, Donald Dietrick, Jurgen Ebbinghaus, and Peter Ladstatter. "Building a Prototype OpenGIS Demonstration from Interoperable GIS Components", *Interoperating Graphic Information Systems Second International Conference, INTEROP'99*: 139-150, 1999.
- (Egenhofer, 1993) Egenhofer, Max J. "What's special about spatial? Database Requirements for vehicle navigation in geographic space", *Proc., ACM SIGMOD Conf.*: 398-402, 1993.
- (ESRI, 2001) ESRI, What is ARCGIS: 2001.
- (Fonseca, 1999) Fonseca, Frederico T. and Max J. Egenhofer. "Knowledge Sharing in Geographic Information Systems", *The Third IEEE International Knowledge and Data Engineering Exchange Workshop*: 85-90, 1999.
- (Mutenda, 2000) Mutenda, Lawrence and Masaru Kitsuregawa. "Parallel R-tree Spatial Join for a Shared-Nothing Architecture," *Database Applications in Non-Traditional Environments (DANTE'99), Proceedings International Symposium on*: 423-430, 2000.
- (NIMA, 1996) NIMA, National Imagery and Mapping Agency. VPFVIEW 2.1 User's Guide: 1996.
- (OGC, 1998) Open GIS Consortium (OGC). OpenGIS Simple Features Specification For SQL, Revision 1.0: http://www.opengis.org/public/sfr1/sfsql_rev_1_0.pdf, 1998.
- (Orlowska, 2000) Orlowska, Maria E. and Xiaofang Zhou. "A Spatial Database as a Component of Integrated Database System," *Database Applications in Non-Traditional Environments (DANTE'99), Proceedings International Symposium on*: 203-212, 2000.
- (Park, 1999) Park, Ho-Hyun, Chan-Gun Lee, Yong-Ju Lee, and Chin-Wan Chung. "Early Separation of Filter and Refinement Steps in Spatial Query Optimization," *Database Systems for Advanced Applications, Proceedings 6th International Conference on*: 161-168, 1999.
- (Rao, 2000) Rao, Ananth, George S. Percivall, and Yonsook Enloe. "Overview of the OGC Catalog Interface Specification," *Geoscience and Remote Sensing Symposium, Proceedings IGARSS 2000, IEEE International*, 3: 1211-1213, 2000.
- (Ravada, 2000) Ravada, Siva and Jayant Sharma. "Trends in Spatial Databases," *Urban Planning and Development Applications of GIS, American Society of Civil Engineers*: 61-77, 2000.
- (Shekhar, 1999) Shekhar, Shashi, Sanjay Chawia, Siva Ravada, Andrew Fetterer, Xuan Liu, and Chang-tien Lu. "Spatial Databases – Accomplishments and Research Needs," *Knowledge and Data Engineering, IEEE Transaction on*, 11/1, 1999.

- (Theodoridis, 1998) Theodoridis, Yannis, Emmanuel Stefanakis, and Timos Sellis. "Cost Models for Join Queries in Spatial Databases", *IEEE Proceedings of ICDE'98*: 1998.
- (Theodoridis, 2000) Theodoridis, Yannis, Emmanuel Stefanakis, and Timos Sellis. "Efficient Cost Models for Spatial Queries Using T-Trees," *Knowledge and Data Engineering, IEEE Transaction on*, 12/1, 2000.
- (Worboys, 1995) Worboys, M. *GIS – A Computing Perspective*. Bristol: Taylor & Francis Inc., 1995.
- (Yin, 2000) Yin, Zhangshi. "Mapping EOS Data on Web," Geoscience and Remote Sensing Symposium, Proceedings IGARSS 2000, IEEE International, 3: 2140-2142, 2000.
- (Zhu, 2000) Zhu, Hongjun, Jianwen Su, and Oscar H. Ibarra. "Toward Spatial Joins for Polygons," *Scientific and Statistical Database Management, Proceedings 12th International Conference on*: 231-244, 2000.

VITA

1st Lt. Engin Colak graduated from Maltepe Military High School in Izmir, Turkey in July 1990. He entered undergraduate studies at the Turkish Air Force Academy in Istanbul, Turkey, where he graduated with a Bachelor of Science degree in Computer Engineering in August 1994.

His first assignment was at Cigli AFB as a student in Undergraduate Pilot Training in August 1994. He graduated as a pilot from Cigli AFB in February 1996 and as a war pilot from Konya AFB in September 1996. His first active assignment was 111th Bomber Squadron at Eskisehir AFB in October 1996. In August 2000, he entered the Air Force Institute of Technology as a graduate student in the computer engineering program.

REPORT DOCUMENTATION PAGE				Form Approved OMB No. 074-0188	
<p>The public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of the collection of information, including suggestions for reducing this burden to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to a penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number.</p> <p>PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.</p>					
1. REPORT DATE (DD-MM-YYYY) 08-03-2002		2. REPORT TYPE Master's Thesis		3. DATES COVERED (From – To) Mar 2001 – Mar 2002	
4. TITLE AND SUBTITLE PORTABLE HIGH-PERFORMANCE INDEXING FOR VECTOR PRODUCT FORMAT SPATIAL DATABASES				5a. CONTRACT NUMBER	
				5b. GRANT NUMBER	
				5c. PROGRAM ELEMENT NUMBER	
6. AUTHOR(S) Colak, Engin, Lieutenant, TUAF				5d. PROJECT NUMBER	
				5e. TASK NUMBER	
				5f. WORK UNIT NUMBER	
7. PERFORMING ORGANIZATION NAMES(S) AND ADDRESS(S) Air Force Institute of Technology Graduate School of Engineering and Management (AFIT/EN) 2950 P. Street, Building 640 WPAFB OH 45433-7765				8. PERFORMING ORGANIZATION REPORT NUMBER AFIT/GCE/ENG/02M-02	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Air Force Research Laboratory /SNZW/AFMC Attn: Mike R. Foster Bldg 620 S1D34, 2241 Avionics Circle WPAFB OH 45433-7303 Comm: (937) 656-4464 DSN: 986-4464 x 3002 Email: mike.foster@wpafb.af.mil				10. SPONSOR/MONITOR'S ACRONYM(S) AFRL/SNZW	
				11. SPONSOR/MONITOR'S REPORT NUMBER(S)	
12. DISTRIBUTION/AVAILABILITY STATEMENT APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.					
13. SUPPLEMENTARY NOTES					
14. ABSTRACT <p>Geo-spatial databases have an overall performance problem because of their complexity and large size. For this reason, many researchers seek new ways to improve the overall performance of geo-spatial databases. Typically, these research efforts are focused on complex indexing structures and query processing methods to capture the relationships between the individual features of fully-functional geo-spatial databases.</p> <p>Visualization applications, such as combat simulators and mission planning tools, suffer from the general performance problems associated with geo-spatial databases. This research focuses on building a high-performance geo-spatial database for visualization applications. The main approach is to simplify the complex data model and to index it with high-performance indexing structures. Complex features are reduced to simple primitives, then indexed using a combination of a disk-based array and B⁺-Trees.</p> <p>Test results show that there is a significant performance improvement gained by the new data model and indexing schema for low to medium zoom levels. For high zoom levels, there is a performance drop due to the indexing schema's overhead.</p>					
15. SUBJECT TERMS Spatial Databases, Geo-spatial Databases, Spatial Indexing, Geo-spatial Indexing, Geographic Information Systems, Vector Product Format, VMAP Database, OpenMap, Multi-dimensional Indexing, High Performance Geo-spatial Indexes					
16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			19b. TELEPHONE NUMBER (Include area code)
U	U	U	UU	123	Maj. Karl S. Mathias (937) 255-6565 x4280