10-24-1996

Semantic geographic information system

Elma L. Alvarez

Florida International University

DOI: 10.25148/etd.FI14031603

Follow this and additional works at: <https://digitalcommons.fiu.edu/etd>

Part of the [Computer Sciences Commons](#)

Recommended Citation

Alvarez, Elma L., "Semantic geographic information system" (1996). *FIU Electronic Theses and Dissertations*. 1262.
<https://digitalcommons.fiu.edu/etd/1262>

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

FLORIDA INTERNATIONAL UNIVERSITY

Miami, Florida

SEMANTIC GEOGRAPHIC INFORMATION SYSTEM

A thesis submitted in partial satisfaction of the

requirements for the degree of

MASTER OF SCIENCE

IN

COMPUTER SCIENCE

by

Elma L. Alvarez

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

This thesis, written by Elma L. Alvarez, and entitled SEMANTIC GEOGRAPHIC INFORMATION SYSTEM, having been approved in respect to style and intellectual content, is referred to you for judgement.

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

David Barton

Wei Sun

Naphtali D. Rishe, Major Professor

Date of Defense: October 24, 1996.

The thesis of Elma L. Alvarez is approved.

Dean Arthur W. Herriott
College of Arts and Sciences

Dr. Richard L. Campbell
Dean of Graduates Studies

Florida International University, 1996

ACKNOWLEDGEMENTS

I wish to thank the members of my committee for their helpful comments and guidance that have made my accomplishments possible.

ABSTRACT OF THE THESIS

Semantic Geographic Information System

by

Elma L. Alvarez

Florida International University, 1996

Miami, Florida.

Professor Naphtali D. Rishe, Major Professor

This thesis research describes the design and implementation of a Semantic Geographic Information System (GIS) and the creation of its spatial database. The database schema is designed and created, and all textual and spatial data are loaded into the database with the help of the Semantic DBMS's Binary Database Interface currently being developed at the FIU's High Performance Database Research Center (HPDRC). A friendly graphical user interface is created together with the other main system's areas: displaying process, data animation, and data retrieval. All these components are tightly integrated to form a novel and practical semantic GIS that has facilitated the interpretation, manipulation, analysis, and display of spatial data like: Ocean Temperature, Ozone(TOMS), and simulated SeaWiFS data. At the same time, this system has played a major role in the testing process of the HPDRC's high performance and efficient parallel Semantic DBMS.

TABLE OF CONTENTS

CHAPTER	PAGE
I. INTRODUCTION	1
- Objective	2
II. BACKGROUND	4
- What is a Geographic Information System (GIS)?	4
- Who uses it ?	4
- Semantic vs. Relational Database Management System (DBMS) ?	5
- Features of the Semantic DBMS used in the GIS implementation	11
III. APPROACH	14
- Implementation Language	14
- Hardware description	14
- Tools used	15
- Environment and Platform implementation	15
IV. SEMANTIC DATABASE	16
- Schema design	16
- Schema Implementation	18
V. SEMANTIC GIS IMPLEMENTATION	20
- User Interface	20
- Display Process	27
- Data Animation	31
- Data Retrieval	32
- Data Transfer	34
VI. TESTING DATA	36
- Semantic	36
- Spatial	37
VII. APPLICATION RESULTS	40
- Advantage	40
- Disadvantage	42
- Data Analysis	42
- Ways to improve it	46
VIII. CONCLUSION	47
- Future work	48
APPENDIX A	50
APPENDIX B	68
BIBLIOGRAPHY	111

LIST OF FIGURES

	PAGE
1. Figure 1	17
2. Figure 2	21
3. Figure 3	21
4. Figure 4	22
5. Figure 5	22
6. Figure 6	26
7. Figure 7	43
8. Figure 8	44
9. Figure 9	44
10. Figure 10	45
11. Figure 11	45

LIST OF TABLES

	PAGE
1. Table 1	7
2. Table 2	10

I. INTRODUCTION

The development of Geographic Information Systems (GIS) is not new. There are some systems such as the Canadian GIS [1] that were developed about 30 years ago. However, there has been a remarkable recent surge of academic, political, commercial, and scientific interest. Many of these initiatives have been stimulated by remote-sensing developments. In particular, the following factors are relevant:

- large quantities of satellite remote sensed data are now available with good radiometric, temporal, and geometric resolution. For example, a terabyte of valuable data per day is projected to be generated by NASA's Earth Observation System (EOS) [2], which will permanently store many petabytes of spatial data[3]. Besides, these data sets are but a small fraction of the information being continually generated at several different research centers.
- advanced computer processing and display technology now make practical the manipulation of large quantities of geographically referenced data for several different types of applications

The need for an efficient GIS is also present at the High Performance Database Research Center (HPDRC) [4] where I work as a research assistant. There, we have a wide variety of spatial data sets from several locations. The spatial data sets include mainly: Ocean Temperature data (supplied by the University of Miami Rosenstiel School of Marine and Atmospheric Science), Simulated SeaWiFS (Sea-viewing Wide Field-of view Sensor) (deployed by NASA), and Ozone data (TOMS, Total Ozone Mapping Spectrometer) (deployed by NASA's Goddard Space Flight Center.) Therefore, there is a need for a computer based system able to efficiently store, manipulate, analyze, and display the information inherent in these data products. And at the same time, there is a

need for a system to test the high performance and efficient massively parallel semantic Database Management System (DBMS) currently being developed at HPDRC. Hence, a semantic GIS is a solution to both needs.

Objective

The purpose of this research is to design and implement a semantic Geographic Information System, which instead of using a relational Database Management System, is going to use a semantic DBMS for its spatial data storage and retrieval. This semantic DBMS will provide protection and security for the data at the same time that it will enforce consistency of the data stored and provide massive retrieval of data to multiple users. The use of a semantic model also ensures better logical properties: friendlier and more intelligent generic user interfaces based on the stored meaning of the data, comprehensive enforcement of integrity constraints, greater flexibility, and substantially shorter application programs [5].

In addition, GISs running on a semantic database, are more efficient than on other database machines[5][6]. The higher efficiency goal can be attained by exploiting the system's understanding of the data's semantics and by achieving a high abstraction level. Besides, the use of the semantic model allows a better exploitation of parallelism by providing a means of distributing data among processors in a way which is invisible to both database programmers and database users.

This GIS is designed to facilitate the interpretation of spatial data like Ocean Temperature, Ozone (TOMS), and SeaWiFS (Ocean color) data and the displaying of those data sets. And at the same time, this system provides an interface between the users and the semantic database that contains the data. This GIS accepts run-time parameters

from the users and uses them to make queries to the database and to retrieve the requested information.

Basically, this semantic GIS will provide efficient storage, manipulation, analysis, representation, and retrieval of both conventional and spatial data. And all of these will be possible by implementing a very friendly graphic user interface and a good set of analysis tools, and by building the GIS using a high performance semantic DBMS. These features are the main factors for the success of the system.

This research also includes the design and implementation of a semantic database. The database schema needs to be designed and created, and all data for the different data types need to be loaded into the database. This semantic database will contain textual and spatial information. Textual information includes satellite and instrument features and observation program description, plus many others. Spatial data is going to include mainly: Ocean Temperature, Ozone (TOMS), and SeaWiFS data. This is a general overview of the GIS's database design and implementation.

II. BACKGROUND

What is a Geographic Information System?

Geographic Information Systems are a combination of computer hardware, software, and procedures designed to support the capture, management, analysis, modeling, and display of geographically referenced data[7]. GIS as a sophisticated computer based mapping and information retrieval system, consists of three primary components:

- A powerful computer graphic program that is used to draw maps, color tables, and others graphic components
- One or more external databases that are linked to the objects shown on the maps. This linkage permits changes entered into the database to be immediately displayed on the maps and querying of the database directly from the map.
- A set of analysis tools that can be used to graphically interpret the externally stored data; for example, by showing objects or regions that need certain criteria in different colors or shadings.

A GIS must have all three components tightly integrated in order to be a good computer system.

Who uses GIS?

Geographical Information Systems technology can be used for scientific investigation, resource management, development planning or any one who needs to interpret or display large quantities of data on a spatial or geographic basis. Frequent GIS's users include:

- Governments: for tracking land usage, utilities, real state parcels, and for emergency response planning in case of natural disasters.
- Businesses: for interpreting demographic data.

- **Scientists:** for depicting the distribution of geological formations, soils, plants, and animals; and for the analysis and study of substance concentration at different atmospheric levels: troposphere, stratosphere, and mesosphere; temperature changes; light reflection, and many others natural changes.

All these people have been using GIS since the 1980's when the first commercial systems became available. But the variety of systems was limited until the advent of powerful and relatively low-cost computers in the late 1980's. Then, there was greater access to specialized software and data from large mainframe computers, so more people started using GISs for several applications. Since then, many others GISs have been developed and people in different work places use them every day.

Semantic vs. Relational Database Management Systems

The main criterion normally used to classify DBMSs is the data model on which the DBMS is based. The data models used most often in current commercial DBMSs are the relational, semantic, network, and hierarchical models. Therefore, DBMSs are categorized as relational, semantic, network, and hierarchical. In this discussion I will only concentrate in the first two models, relational and semantic.

The relational model represents the database as a collection of relations, which are also called tables. When a relation is thought of as a table of values, each row in the table represent a collection of related data values. These values can be interpreted as facts describing a real-world entity or a relationship[8]. There are also two types of tables: tables representing the application's objects and tables representing relationship between objects. Each object table consist of at least one column (called unique key) that identifies the objects, and several others columns that display data about the objects[9]. In addition,

some concepts in the relational database model are: tuples (a row), domain (data type describing the types of values that can appear in each column).

The best known relational database language is the Structure Query Language (SQL), which is a comprehensive database language. It has statements for data definition, query and update. Hence, it is both a Data Definition Language (DDL) and a Data Manipulation Language (DML). In addition, it has facilities for defining views on the database and for creating and dropping indexes on the files that represent relations[10].

The following table is a representation of a simple relational database instance of the spatial database schema used for the GIS. This table depicts many of the concept described above.

DATA TYPE			
id-key	data-name	byte-number	has-table-id-key
12356-	Ozone	576-	12458-

TAKE-BY	
id-key	satellite-name-key
12356-	Nimbus-7
12356-	Meteor-3

COLOR-TABLE			
id-key	color-list	value-list	description
12458-	1, 2, 3,	256 200, 210,	560 Ozone layer in Dobson Unit

SATELLITE	
satellite-name-key	instrument-name
Nimbus-7	N-7 TOMS
Meteor-3	M-3 TOMS

INSTRUMENT	
instrument-name	ohser-prog-id-key
N-7 TOMS	12568-
N-3 TOMS	12570-

OBSERVATION-PROGRAM					
ohser-prog-id-key	frequency	start-date	end-date	description	map-id-key
12568-	Monthly	110178-	040194-	Nimbus-7 Ozone	12700-
12568-	Daily	010191-	050593-	Nimbus-7 Ozone	12701-
12570-	Monthly	080191-	120194-	Meteor-3 Ozone	12702-
12570-	Daily	082291-	122794-	Meteor-3 Ozone	12703-

BITMAP		
map-id-key	real-date	image
12700-	030192-	picture
12701-	110793-	picture
12702-	070194-	picture
12703-	060892-	picture

Table 1. Relational database instance of the GIS's spatial database

The relational database provides a good service in many conventional database applications. And it has a powerful and easy to use user language (SQL) for retrieval and update of databases. However, in situations where the structure of information is complex, or where greater flexibility is required or where non-conventional data is involved, other approaches need to be considered.

One good approach to consider is the semantic database model. The central notion of the semantic model is the concept of an object, which is any real-world item, and it can be either a concrete object or an abstract object[9]. These objects are categorized into classes according to their common properties. These classes, called categories, can be disjoint or

not disjoint depending of the object that belongs to the categories under analysis. The representation of the objects in the computer is invisible to the users, who perceive the objects as real-world entities, which can be tangible like persons, cars, or telephones, or intangible such as ideas, observations, or desires [5].

The semantic database is perceived by its users as a set of facts about objects. And these facts can be of the following types:

- facts stating that an object belongs to a category. This is also called unary fact (Object-category). Unary facts categorize objects of the real world.
- facts stating that there is a relationship between objects. This is also called a binary facts (Object-Relation-Object). Binary facts establish relationships of various kinds between pairs of objects.
- facts relating objects to data such as dates, images, or analytical functions[5].

Some of the semantic languages are: a fourth-generation structured extension of a structure third-generation programming language (Ex. Pascal), and a non-procedural predicate calculus language.

Although the semantic data models are somewhat different in their terminology and their selection of tools used to describe the semantics of the real world, they have several common principles:

- The entities of the real world are represented in the database in a manner invisible to the user. On the contrary, in the relational model, the entities are represented by the values of the keys of some tables. Then, the user-invisible representations of real world entities are referred to as “abstract objects”. The “concrete objects”, or “printable values,” are numbers, characters, etc. [10].
- Logically-explicit relationships are specified among abstract objects and between abstract objects. There are no direct relationships among concrete objects. And in

most semantic models, only binary relations are allowed, since higher order relations do not add any power of semantic expressiveness[11][12], but do decrease the flexibility of the database and the representability of partially-unknown information, further they add complexity and the potential for logical redundancy [10][11].

The following table shows a semantic database instance of the spatial database used by the GIS. This instantaneous database represents the same state as the relational instantaneous database of Table 1. And if you compare these two tables, you can better see the structural differences between the relational and the semantic approaches.

category	object #	relation	object # or value
BITMAP	01 -	real-date	030192 -
	01 -	image	picture
	02 -	real-date	110793 -
	02 -	image	picture
	03 -	real-date	070194 -
	03 -	image	picture
	04 -	real-date	060892 -
	04 -	image	picture
Observation-program	05 -	frequency	daily
	05 -	start-date	010191 -
	05 -	end-date	050593 -
	05 -	description	Nimbus-7 Daily ozone
	05 -	has-data	01 -
	06 -	frequency	monthly
	06 -	start-date	110178 -
	06 -	end-date	040193 -
	06 -	description	Nimbus-7 Monthly ozone
	06 -	has-data	02 -

Table 2. A semantic database instance for the GIS application

category	object #	relation	object # or value
Observation-program	07 -	frequency	daily
	07 -	start-date	082291 -
	07 -	end-date	122794 -
	07 -	description	Meteor-3 daily ozone
	07 -	has-data	03 -
	08 -	frequency	monthly
	08 -	start-date	080191 -
	08 -	end-date	120194 -
	08 -	description	Meteor-3 Monthly ozone
	08 -	has-data	04 -
Instrument	09 -	instrument-name	N-7 TOMS
	09 -	obs-prog-used	05 -
	09 -	obs-prog-used	06 -
	10 -	instrument-name	M-3 TOMS
	10 -	obs-prog-used	07 -
	10 -	obs-prog-used	08 -
Satellite	11 -	satellite-name	Nimbus-7
	11 -	instrument-used	09 -
	12 -	satellite-name	Meteor-3
	12 -	instrument-name	10 -
Color-Table	13 -	color	1, 2, 3, 256
	13 -	value	200, 210, 560
	13 -	description	Ozone Layer in Dobson unit
Data Type	14 -	data-name	Ozone
	14 -	number-of-bytes	576 bytes
	14 -	take-by	11 -
	14 -	take-by	12 -
	14 -	has-table	13 -

Table 2. Continued

In general, the semantic DBMS systems represent the information by logical associations (relations) between pairs of objects and by classification of objects into categories. Hence, the semantic models offer a simple, natural, flexible and non-redundant specification of information[9]. In the semantic models, as the word implies, the meaning

of users' information is captured and a concise, high level description of the information is provided.

In addition, the semantic models have a potential for a more efficient implementation than the relational models for the following reasons:

- all the physical aspects of representation of information by data are invisible to the users in the semantic models. This creates a greater potential for optimization: more things can be changed for efficiency considerations without affecting the user programs. The relational model has more data independence than other models; for example, the order of rows in the tables is invisible to the users. But the semantic models have even more data independence; for example, the representation of real-world entities by printable values is invisible to the users[10].
- In the semantic models the system knows more about the meaning of the user's data and about the meaningful connections between such data. This knowledge can be utilized to organize the data so that meaningful operations can be performed faster at the expense of less meaningful or meaningless operations[10].

Features of the Semantic DBMS used in the GIS's implementation

GISs demand high performance and pose some very special requirements for database management. Hence, DBMSs designed for commercial usage are not well suited for GIS because they generally lack the special provision to achieve physical clustering necessary for accommodating spatial data and for coping with fast access to spatial data [13][14]. However, the semantic DBMS that is going to be used in the GIS implementation has some features that make it a high performance and efficient massively parallel semantic DBMS that doesn't contain the previously mentioned deficiencies. This system is useful for most typical database applications as well as specialized domains such as Earth Science.

This DBMS provides highly efficient accumulation and retrieval of general, scientific and spatial data.

This is a semantic DBMS that has been developed with a semantic/object oriented approach, so this system satisfies the three essentials needs of many database applications:

- strong semantics embedded in the database to handle the complexity of information.
- storage of multi-dimensional spatial, image, scientific, and other non-conventional data.
- very high performance to allow massive data flow [15].

This object-oriented system is based on the semantic binary model of databases and has good logical properties such as:

- friendlier and more intelligent generic user interfaces based on the stored meaning of the data.
- comprehensive enforcement of integrity constraints
- greater flexibility - since objects are not required to be identified by any key
- substantially shorter application programs[15]
- higher efficiency for both small and massive numbers of processors [15]
- better exploitation of parallelism for data storage and processing [15].

The semantic parallel architecture of this database system also provides efficient and flexible access to a large collection of data stored on various physical devices (memory, disk, and tertiary storage). In addition, data reference transparency is an inherent property of this semantic binary model system. A semantic binary database system addresses any object directly by its content, using an efficient database organization[10]. Another good feature of this semantic DBMS is that in this database, large mapping from objects to their physical location does not have to be kept. Each host interface in the system has a

small allocation map containing highly compressed data references for each computer storage unit.

III. APROACH

Implementation Language

This system was implemented in C++ because of the good features provided by this language. C++ is a small, block structured programming language with only 48 commands (keywords), so complicated problems can be solved with small amount of code. In addition, C++ has a rich operator set, and a nearly orthogonal design, and it is a terse and extensible language. Besides, C++ is a highly portable language and translators for it are available on many different machines and systems. Also, C++ has all the features of a good Object Oriented Programming Language (OOP).

However, in order to enhance the GIS and make it even more small, simple, and portable across several platforms. I am developing a version of the system in Java. One main advantage of the Java language over other programming languages is its portability via the Word Wide Web (WWW). Java applets can be used to create animation, figures, menu options that can respond to inputs from the users, games, or other interactive effects on the same Web page among the text and graphics. At the same time, Java applications can run on any machine as a C++ program does.

Hardware description

The main computer used to compile, test, and run the GIS system is a SPARC Server-10 machine. This is a Sun work station with 96 MB of RAM (Random Access Memory). In addition, this machine has 2 RISC based processors with 50 MHz each that give enough power to the computer to comfortable run any UNIX base program. For future reference the name of this machine is Dizzy.

Tools used

In order to make a good graphical representation of the user interface and map displaying, a graphical tool was used. The main software used for the graphics was Simple Raster Graphics Package (SRGP). This package is composed of several library functions that were used depending of the need: color loading, buttons, pull-down menus, top-bar menu, tables, mouse movement tracking, map frames display and map formats. This is not a very flexible software to work with. This is a kind of primitive software, and most of the graphical representations require a lot of work from the programmer.

Environment and Platform implementation

This system was implemented in the UNIX environment and uses the X-Windows system to manage the display and the windows' management. X-windows system allows to open several windows of different sizes and allows to have control of the commands inside the window. This was of great help in the development of the GIS since in the system, several windows are opened specially for the zoom option, and all those windows are interactive to allow the users to manipulate the data. In addition, the X-windows system allows the users to capture the images currently displayed on the screen, so the program can have knowledge of what is being display and use this information to better superimpose the images.

IV. SEMANTIC DATABASE

A semantic binary database schema is a set of categories, relations, and database types. A category is a specification for database abstract objects that belong to this category. Each category may have several relations with other categories and data types. A relation from a category to a data type is called an attribute. A relation from a category to a category is called an abstract relation. But, since these two relations have many common features, the Semantic Binary Database Interface gives only one command 'NewRelation' to create both type of relations. In the category cases, the 'NewCategory' command is used to create the new categories.

Schema design

The spatial database that it is going to be used for the GIS has some specific requirements. This database has to store spatial data together with semantic data on the same database. Some GIS systems store the spatial data separately from the semantic data [16], which make the system inefficient and difficult to use [9]. This GIS's database also has to store several information for many different data sets, and must cover the following information for each data set:

- name and description of the satellite used.
- name and description of the instrument used for reading.
- color table containing: color, value, and a short description of the table information.
- observation program used, including: data frequency, start date and end date of the readings, and a short description of the data collected.
- collection of all binary data files containing the data maps for each day, week or month depending on the data set.
- date for each file or map stored.

Trying to follow all these requirements, the following semantic database schema was designed:

Figure 1. Semantic schema for the GIS's database

This schema contains six abstract categories and several relations, which include abstract relations and attributes, that make possible the storage and retrieval of all information from the database.

Schema Implementation

For the schema implementation, I developed a program that creates and loads all data into the database. This program was written in C++, since the Semantic Binary Database Interface was implemented in C++. This was a relative short program with very few transactions. The following line of code creates a database called "Earth.DB"

```
TDataBase *DB = CreateDataBase("Earth.DB");
```

and the following is a listing of the method 'CreateSchema', which as the name says creates the semantic database schema described in Figure 1. Functions used in this method are provided by the Semantic Binary Database Interface.

```
void
CreateSchema(TDataBase *DB)
{
    do{
        DB->Transaction_Begin();
        NewCategory(DB, "DataType");
        NewRelation(DB, "dname", "DataType", "String");

        NewCategory(DB, "ColorTable");
        NewRelation(DB, "color", "ColorTable", "Binary");
        NewRelation(DB, "value", "ColorTable", "Binary");
        NewRelation(DB, "mapcolor", "ColorTable", "Binary");
        NewRelation(DB, "description", "ColorTable", "String");
        NewRelation(DB, "hastable", "DataType", "ColorTable");

        NewCategory(DB, "Satellite");
        NewRelation(DB, "sname", "Satellite", "String");
        NewRelation(DB, "takeby", "DataType", "Satellite");
    }
```

```

NewCategory(DB, "Instrument");
NewRelation(DB, "iname", "Instrument", "String");
NewRelation(DB, "uses", "Satellite", "Instrument");

NewCategory(DB, "ObserProg");
NewRelation(DB, "frequency", "ObserProg", "String");
NewRelation(DB, "startdate", "ObserProg", "Integer");
NewRelation(DB, "enddate", "ObserProg", "Integer");
NewRelation(DB, "description", "ObserProg", "String");
NewRelation(DB, "progused", "Instrument", "ObserProg");

NewCategory(DB, "BitMap");
NewRelation(DB, "realdate", "BitMap", "Integer");
NewRelation(DB, "image", "BitMap", "Binary");
NewRelation(DB, "date", "BitMap", "String");
NewRelation(DB, "hasdata", "ObserProg", "BitMap");
}
while(DB->Transaction_End() == ec_concurrency);
}

```

The program also consists of several other methods to load all the information into the database. For instance, the function 'LoadImage' loads all the binary data files into the database, and the method 'LoadConts' loads two files, one that contains the continents and the other one that contains the edges or outline of the continents. In addition, the method 'LoadData' loads most of the textual data into the database. A listing of these methods and a complete listing of the source code is depicted on Appendix A.

V. SEMANTIC GIS IMPLEMENTATION

This GIS is completed independent of the data upon which it operates. It retrieves all the information (data) from the semantic database. This makes the GIS a generic system that works for a variety of data sets.

User Interface

I tried to develop a very friendly graphical user interface that portrays flexibility, simplicity, and high data manipulation techniques. This system interacts with the users using two input devices: keyboard and mouse. Hence, users have the flexibility of selecting any menu option using their favorite device. The main features of the GIS's interface are top bar, drop-down menu, buttons, and icons. All these combined allow the user to easily manipulate the data, and navigate through the semantic database schema and retrieve the desire information currently stored. The menu options include: Projection, Data, Satellite, Instrument, Frequency, Rotation, Date, Cloud, Zoom, and Exit.

- Projection - allows the users to choose whatever projection they desire to see the data on. Possible choices are: Orthographic, Orthogonal, Sinusoidal, Stereographic, Homolographic, and Mercator. Each of these projections portrays the data from a different view point. For instance, in the Stereographic projection, the whole map is displayed in a circular format, but some mathematical computation are applied to each coordinate point, so both Poles (North and South) can be view in the same frame. And in the Orthogonal and Orthographic the data point maintain their original coordinates values, but with the difference that the Orthographic projection portrays the data in a sphere format, and the Orthogonal portrays the data in a rectangular format The following figure (Figure 2) depicts an illustrated projection sequence of ozone data maps in the same order that the projections were mention above:

Figure 2. Ozone data maps in different projections

- **Data** - once this menu option is activated a query is done to the database requesting the name of all the data sets currently stored there. Then, the result of the query is presented to the users as a list of possible options. The data sets that are currently available on the database include: Ocean Temperature, Ozone, and simulated SeaWiFS data. Figure 3 shows a sample of the Ocean Temperature data depicted in a Sinusoidal projection. Figure 4 shows a sample of the Ozone data in the Homolographic projection, and Figure 5 depicts a sample of the simulated SeaWiFS data in an Orthogonal projection.

Figure 3. Weekly Ocean Temperature data for 1987

Figure 4. Nimbus-7 Daily Ozone (TOMS) data for 1992

Figure 5. Simulated SeaWiFS data for 1994

- **Satellite** - once the user clicks on this menu option, a query is performed to the database requesting a list of all recorded satellite names for the already chosen data set. Then, this list of possible choices is presented to the users, so they can select one name. For example, ozone data is provided from two different satellites: Nimbus 7 and Meteor 3. However, for the SeaWiFS data there is just one choice: SeaStar.
- **Instrument** - once the users activate this menu, a query is done to the database requesting the name of all instruments which were used in the already selected satellite. Then, the result of the query is presented to the users as a list from which they can select one name. For instance, possible instrument's name for the Nimbus 7 satellite is TOMS, and for the SeaStar satellite, there is a list of eight possible sensors.
- **Frequency** - once this menu option is activated, a query is formulated to the database requesting all recorded frequencies for the already chosen satellite and instrument. Then, the list of available frequencies is presented to the users allowing them to select a desired frequency. Possible frequencies are: daily, weekly, and monthly depending on the data set.
- **Rotation** - allows the users to Reduce, Stop, or Increase the rotation speed of the currently displayed frames. Every time that the users click on the Increase option, the rotation of the image is increased by a factor of two. Then, they can use the Reduce option to gradually reduce the speed by a factor of two or just activate the Stop menu and stop the rotation completed with only one mouse click.
- **Date** - every time that a new frame is displayed on the screen, the corresponding measured date is also displayed on the south part of the screen. So among the options provided by this menu are: Single Frame Forward, Single Frame Backward, Reverse, Stop, and Increase the date:

~ Single Frame Forward: once this menu is activated, the data is displayed one date at a time as opposed to continuously displaying data frames. Every time the users click with the mouse on this menu option, a new frame is displayed on the screen. So, until the menu is activated again the same frame is static on the screen. This menu allows users to observe and analyze a particular data set for an specific satellite, instrument and date for as long as they wants. Then, click on the menu again to view the frame for the next date.

~ Single Frame Backward: this option allows to display one date at a time, but going back on time. So if the frequency is daily, activating this option allows the user to view the frame belonging to the previous day. Hence, if the client forgot to observe any detail, he or she can go back a view that particular frame again. In addition, this option provides all the advantage of the Single Frame forward.

~ Increase: by default, the program runs on this mode. When this menu is active the data is displayed as a continuous movie (one frame after the other one) while incrementing the date.

~ Reverse: when this menu is activated, the data is displayed as a continuous movie going back on time. Hence, at any time while viewing the movie, users can click on this menu and then they will start viewing the previously display frames again. For example, if a monthly data is being display and ... June, July, and August were just displayed before the menu was activated, then after the activation, the display sequence will continue, but it will display August, July, June,

~ Stop: this menu allows to stop the animated movie at the date currently being displayed. Thus, users can stop in any particular frame of their interest to better view details of the data set currently being display. This menu option combined with Increase, and Reverse allows the users to view a movie for an specific range of time

over and over again, going back and forth and stopping on a desired date until all needed details of the data are collected and analyzed.

- Zoom - allows the users to zoom in or out a particular geographical location by clicking with the mouse on a start point and dragging the mouse until a desired end point. Then, the coordinates of the chosen area are recorded into the program and a new window of the selected frame size is opened displaying a movie of that piece of data zoom out to the chosen percent along with the date. Possible percentages range from 20% to 120%. This menu also allows the users to go back to the main program at any time by just clicking the mouse. Some of the other advantage of this menu option is that users can view in more detail a particular geographic area of interest since they can zoom in or out as desire. Besides, the screen can hold at the same time the complete main program window interface plus all the windows containing the selected zoomed frames. This gives a great observation and analysis capability to the users since the same and different geographical locations can be viewed on several windows at the same time and at different zooming percent and different projections. Figure 6 portrays a sample of a screen after several areas have been zoom out from the main Ocean Temperature data frame.

Figure 6. Ocean Temperature data zoom out at several percent

- Cloud - offers the users two choices: Visible and Non-Visible Clouds.
 - ~ Visible: In the Ocean Temperature data, this menu allows to see the clouds coverage that was recorded on the day that the observation were made. In the Ozone data, this allows the users to see the zones where no data was recorded because of instrument mal function. In the SeaWiFS data, this option is not valid. This menu is good for data precision since every bit of data is displayed as recorded.
 - ~ Non-Visible: When this menu is active, in the Ocean Temperature data, the frames are displayed with out any visible clouds. This is done by substitution. Cloud covered

pixels are substituted by previously recorded values before the frame is displayed on the screen. In order to do this, the program has to “remember” where there are clouds and where not. Hence, when the user choose any of these menu options, the right action can be taken. In the Ozone data, this option plays almost the same role as in the Ocean Temperature data, but here the last ozone reading is displayed on the zones where no data was recorded. In this case, instrument failures are treated as cloud coverage. One good thing about these menu options is that users can view the cloud cover and lack of data areas for data precession, or they can omit these for data clarity.

- Exit - allows the user to properly terminate the program by closing all windows and databases opened during the program execution.

Another feature of the GIS's interface is the color table. Each data set has a different color chat to aid in the interpretation of the colors. Thus, once the user selects one particular data set, a color table (color, value) corresponding to that data set is retrieved from the database and displayed on the screen next to the map frame. For the Ocean Temperature data, values are given in degrees Celsius, and for Ozone, values are in Dobson Units (DU). Dobson units indicate the thickness of the ozone layer if it were measured at 0 degrees Celsius and at standard atmospheric pressure (1 atm)[17][18]. More specifically, 0.01mm thickness of ozone at 0 degrees Celsius and at standard atmospheric pressure is defined to be 1 DU.

Display Process

This GIS displays the spatial data image in two modes: static and dynamic. This section is going to concentrate mainly on the static displaying process, and then in the next section, I will discuss the dynamic process and analysis.

A data file for one particular date is first retrieved from the database and placed in a buffer. Then, this buffer is passed to a method where the whole data file is processed differently depending on the data type and the projection. For instance, if the active projection is Homolographic, the following mathematical computation needs to be performed before start processing values from the buffer:

```

xi = XGetImage(d_dpy, d_act.drawable.xid, hpw, 220, Dw, Dh, AllPlanes, ZPixmap);
for(int y = 0; y < Ph; y++)
    for(int x = 0; x < Pw; x++)
    {
        cx = R * (1 - ssin[y] * ccos[x] / (1 + ssin[y] * ssin[x]));
        cy = R * (1 - ccos[y] / (1 + ssin[y] * ssin[x]));
        if((f * (cx + cy * Dw)) < (Dw * Dh))
        {
            map[x + Pw * y] = &xi->data[f * (cx + cy * Dw)];
        }
    }
}

```

The set of all mathematical computations for all projections is placed on a switch or case statement, so whenever a new projection is activated the right case option is executed. The complete listing of the function 'Startup' that contains all these computations can be seen on Appendix B.

Once the computations for the projections are done, the process of the file buffer begins on a method called 'Next_Frame'. In this method the following computations are executed: an x number of bytes are read at a time from the buffer depending on the data type, and then byte per byte is processed, a color is assigned to each byte read, the pixels' coordinates are recorded, and if the Non-Visible Cloud option is active, then for every byte read, the program checks if that coordinate belong to a cloud or to an instrument failure, and the proper action is taken. In order to do this last computation, the program needs to "remember" the location of cloud cover or instruments' failure depending on the data type.

In addition, this GIS is able to “remember” the continent location and the continents’ outline when the Ozone data is being display. The frame containing the outline of the continents is imposed over the Ozone data frame. In order to achieve the displaying of two different frames at the same time, the following process need to be computed: both data (ozone and continent data) are retrieved from the database, and whenever a byte is read from the ozone data buffer, the program checks if the coordinates belong to the geographical location of a continent, and if the answer is ‘yes’, the right computation is performed. Hence, when finally the frame is displayed on the screen users view the Ozone data plus the continents outline all in one frame. This two frame combination was done to help the users to have a better geographical orientation of the ozone layers value and the worldsite.

The following is a fragment of the ‘Next_Frame’ method that performs most of the previously mention computations:

```
for(int y = 0; y < 180; y++)
{
    int n = fastread( b, bts, 1 );
    for(int x = 0; x < 360; x++)
    {
        if(bts == 360)
        {
            n = r[x];
            yy = y;
            tp = color[b[n]];
        }
        else
        {
            n = (2 * ((4 * r[x]) / 5));
            yy = 179 - y;
            if(cmap[r[x] + 360 * yy] == '0')
                tp = color[256 * b[n] + b[n+1]];
            else
                tp = 177;
```

```

}
if((cloud * tp) == 42)
    tp = pcol[or[x] + 360 * yy];
if((bts == 360) && (exts[0] == 'c') && {cloud == 0})
{
    if((tp == 42) && (cmap[r[x] + 360 * y] == '0'))
    {
        tp = 50;
    }
}
tmpm[x] = tp;
ff = map[x + Pw * yy];
switch(opt)
{
    case 0 : // Mercator
        if(( ff != 0) && (tp != *ff))
        {
            if( bts == 576 || exts[0] == 'c' )
            {
                for(int j = 0; j < f; j++)
                {
                    for(int k = 0; k < g1[yy]; k++)
                        *(ff + k * Dw + j) = tp;
                }
            }
            else
            {
                for( int j = 0; j < k; j++)
                {
                    for(int k = 0; k < f*g0[yy]; k++)
                        *(ff + k * Dw + j) = tp;
                }
            }
        }
        break;
    default :
        if(( ff != 0) && (tp != *ff))
        {
            for(int j = 0; j < f; j++)
            {
                for(int k = 0; k < f; k++)
                    *(ff + k + Dw + j) = tp;
            }
        }
}

```

```

        }
    }
}
for(x = 0; x < 360; x++)
    pcol[x + 360 * yy] = tmpm[x];
}

```

When all this required calculations and checks are done, the following line is executed to send the new frame to the screen:

```
XPutImage(d_dpy, d_act.drawable.xid, d_act.gc_frame, xi, 0, 0, hpw, 220, Dw, Dh);
```

As can be seen, the algorithms executed on the 'Next_Frame' method does not run in linear time, so this time has a high weight on the total time that takes to display frames in the screen.

Data Animation

During the data animation process, more computation need to be performed in order to achieve a dynamic movie of all the data sets. If the Date option (Increase) and the Rotation option (Increase) is active, additional mathematical calculations need to be done on the 'Next_Frame' method in order to rotate the whole image at the same time that the new frame is processed and displayed. Two time functions need to be executed at the same time to produce the combined result. Therefore, the resulting output is a sequence of several rotating frames portraying data for nine or ten years depending on the data set. In order to achieve the above mention output, the following lines of code need to be added at the beginning of the 'Next_Frame' method as part of the additional computation needed:

```

for( int x = 0; x < 360; x++)
{
    r[x] = (x + fn + 360) % 360;
    or[x] = (x + Rot + 360) % 360;
}

```

```

}
fn = (fn + Rot + 360) % 360;

```

This UNIX base GIS has a limited movie speed mainly because of the use of the X-Windows system. But the database retrieval time (very small) and the computational time require to process the data before send it to the screen (exponential) also play a role in the movie speed. Therefore, the resulting animated movie have a speed of about five frames per second. Not a very acceptable high speed for a good movie display.

Data Retrieval

This GIS is constantly retrieving information from the semantic database. For instance, when the program is running in the dynamic mode (animated movie), a request is made to the database every few milliseconds to retrieve the date and binary data values. But how and when is the data retrieved from the database?

Every time that a menu option is activated a request is performed to the database requesting the particular set of information. For example, when the satellite option is activated the following source code is executed to retrieve the data from the database and display it on the screen:

```

Var S = RangeQuery(TheRelation("DATATYPE::dname", DB), dtname);
SetQuery Satel = S(TheRelation("DATATYPE::takeby", DB);
while( Satel.GetVarInc(sateln))
{
    Var SName = sateln["Satellite::sname"];
    sprintf(satel_name[i++], "%s", pChar(SName));
    SRGP_text(SRGP_defPoint(xl + 10, yl), satel_name[i-1]);
    yl += 20;
}

```

Retrieving data to get a list of possible menu options is very simple as you have seen in the previous source code listing. However, when a different data set is selected, and the

user has already chosen all the options for a particular satellite, instrument, and frequency that he or she desires to view, more complicated queries are done to the database to extract the right information and start the displaying process. The following is a fragment of the method that perform some of these queries:

```

Var Info = RangeQuery(TheRelation("DATATYPE::dname", DB), dname[dindex]);
Var ext = Info["DATATYPE::cmext"];
Var bt = Info["DATATYPE::bytes"];
Var table = Info["DATATYPE::hastable"];
Var mapcols = table["ColorTable::mapcolor"];
Fptr = 0;
dbread( FileBuff, 1, sizeof(FileBuff), mapcols);
fread(color, 1304, 1);
dbclose(mapcols);
Color_Chart(dname[dindex]);
sprintf(datekey, "%li%s", ind, pChar(xet));
Var Image = RangeQuery(TheRelation("BITMAP::dname", DB), datekey);
SetQuery Obser = instru(TheRelation("Instrument::progused", DB));
while(Obser.GetVarInc(observe))
{
    Var fq = observe["ObserveProg::frequency"];
    if(strcmp(pChar(fq), freqs[findex]) == 0)
    {
        Var prgdesc = observe["ObserveProg::datadescr"];
        Var sdate = observe["ObserveProg::startdate"];
        Var edate = observe["ObserveProg::enddate"];
        break;
    }
}
sprintf(pdscr, "%s", pChar(prgdesc));
SRGP_text(SRGP_defPoint(30, 445), pdscr);
SetQuery bdata = observe(TheRelation("ObserveProg::hasdata", DB));
while(bdata.GetVarInc(Image))
{
    rd = Image["BITMAP::realdte"]
    Rdates[range++] = Long(rd);
}
datesort.InsertionSort(Rdates, range);
for(int indrd = 0; indrd < range; indrd++)

```

```

{
    SetQuery dataf = obserp(TheRelation("ObserProg::hasdata", DB));
    while(dataf.GetVarInc(Image) && check == 0)
    {
        Rdate = Image["BITMAP::realdate"];
        if(Long(Rdate) == Rdates[indrd]);
        {
            in_pt = Image["BITMAP::image"];
            check = 1;
        }
    }
    Fptr = 0;
    dbread(FileBuff, 1, sizeof(FileBuff), in_pt);
    Next_Frame(Long(Rdate), in_pt, Long(bt), pChar(ext));
    Get_action(Long(bt), Rdates, pChar(ext));
}

```

The complete source code listing for all the data retrieval can be seen on Appendix B together with the complete source code listing of the GIS application.

Data Transfer

The Geographic Information Systems and the databases usually run on different machines. In this case, the database (server) is stored in a computer called Miami, and the GIS (client) can run on several machines like Dizzy, n5, or any other machine selected by the user. This is a typical homogeneous system [19] which has a single database stored centrally at Miami, and several clients' machines spread through out the network from which the database can be accessed. Therefore, data needs to be transferred from Miami to the end users' computer. And in order to achieve a good transfer rate, a high speed data transfer device is needed.

For the GIS's testing, I run the system on Dizzy and a ForeRunner ASX-200 switch based on ATM (Asynchronous Transfer Mode) technology is used for the data transfer. This ASX-200 switch delivers high-performance ATM connectivity for LAN (Local Area Network) work group, LAN backbones, and WAN (Wide Area Network) access. In

addition, this ForeRunner switch supports from 2 to 96 connections, and it has an interface speed from 1.5 Mbytes/sec to 155 Mbytes/sec. Besides, it provides 2.5 Gbytes/sec to 10 Gbytes/sec of switching capacity. Moreover, this switch has a 2.5 Gbytes/sec non-blocking architecture, up to 13 312 cell/port buffer, and 3 server levels and 127 sub levels. This switch also offers advanced connection and bandwidth management capabilities. Therefore, this ATM switch plays an important role to achieve fast data transfer between the database's computer (Miami) and Dizzy, the main computer used for testing.

VI. TESTING DATA

Several megabytes of data including semantic and spatial were used to test both the GIS and the Semantic DBMS currently been developed at HPDRC.

Semantic

Semantic data includes mainly information about the data sets, color tables, satellites, instruments, and observation programs used. More specifically, the database currently stores the following textual information:

- name for each data set.
- number of byte per data file.
- color table for each data type including: list of colors and values and a short description.
- Satellite name.
- Instrument name
- frequency of each data sets
- start and end date of each data sets
- description of the displaying data
- date of each data frame display

The GIS does not just displays spatial data. It also gives the user several information about the data currently displays on the screen. And in order to achieve this, all the previously mention textual information is combined with the spatial data on the same database. This is needed to better meet all request coming from the users, and allows them to have information about the spatial data at the same time that they are looking at the images.

Spatial

The spatial data currently store in the database is mainly in a raster format [20] and includes:

- One year of weekly Ocean Temperature data for 1987. This represents 52 data files of 64800 bytes each. Those files are organized in a way that each files have 180 lines of 360 bytes each, and the first bytes read belong to the North Pole latitudes points. In order to process the files, one byte at a time is read, then a formula is applied to it to get the temperature in degrees Celsius. Then, a different color is assigned to each degrees. The following are the first few lines of a data file corresponding to the first week of January 1987:

```
0000000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
0000020 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
0000040 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
0000060 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
0000100 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
0000120 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
.....
```

As can be seen, most of the values are 0's since these temperature values correspond to the North Pole latitudes during January. If the previous lines are compared with the followings:

```
0000000 012 012 012 012 012 012 012 012 012 012 012 012 012 012 012 012
0000020 012 012 012 012 012 012 012 012 012 012 012 012 012 012 012 012
0000040 012 012 012 012 012 012 012 012 012 012 012 012 012 012 012 012
0000060 012 012 012 012 012 012 012 012 012 012 012 012 012 012 012 012
0000100 012 012 012 012 012 012 012 012 012 012 012 012 012 012 012 012
0000120 012 012 012 012 012 012 012 012 012 012 012 012 012 012 012 012
0000140 012 012 012 012 012 012 012 012 012 012 012 012 012 012 012 012
0000160 012 012 012 012 012 012 012 012 012 012 012 012 012 012 012 012
0000200 010 010 010 010 010 010 010 010 010 010 010 010 010 010 010 010
0000220 010 010 010 010 010 010 010 010 010 010 010 010 010 010 010 010
0000240 010 010 010 010 010 010 010 010 010 010 010 010 010 010 010 .....
```

there is a big different in values since this data correspond to the second week of August. The temperature of the water on the North Pole latitudes are more warmer during the summer season.

- Several megabytes of simulated SeaWiFS data that belongs to one day reading of the SeaStar satellite using eight different sensors to capture global ocean color [21]. For each sensor there is a huge data file, so data handling becomes kind of complicated. Therefore, for the GIS and DBMS testing purpose, the data files were simplified, and a new 64800 bytes data files were created. These new files are the ones stored in the database and displayed by the GIS. The resulting image is the same as the original, but some data accuracy is missing.
- Ozone data for a period of up to 15 years (1978-1994), two different satellites and two different frequencies (monthly and daily). The monthly reading are obtained by taking the daily readings for the entire month and recording the mean for each zone. These files are approximately 103680 bytes each. Ozone data are gridded into 1 degree latitude by 1.25 degrees longitude zones. Latitudes go from -90 degrees (the South Pole) to 0 degrees (the equator) to +90 degrees (the North Pole) in 1 degree steps, so there are 180 latitude zones. The first zone extends from -90 to -89, so the grid cell is considered to be centered on -89.5. Similarly, longitudes go from -180 (west longitude) to 0 (Greenwich, England) to +180 (east longitude) in 1.25 degrees steps, so there are 288 longitude zones (360/1.25). The first longitude zone extends from -180 to -178.75, so the grid cell is considered to be centered on -179.375.

As an example, the first few lines of a data file may look like this:

Day: 182	June 30, 1992	Real Time Meteor-3 TOMS	LECT: 12:00 PM
Longitudes:	288 bins centered on 179.375 W to 179.375 E	(1.25 degree steps)	
Latitudes	180 bins centered on 89.5 S to 89.5 N	(1.00 degree steps)	

VII. APPLICATION RESULT

Advantage

This Geographic Information System has facilitated the manipulation, study, analysis, and interpretation of several spatial data sets, and this has been achieved by providing high data manipulation techniques, flexibility, high observation capability, fully documented images, and database navigator interface.

- This GIS provides the users with a high data manipulation techniques that allow them to analyze, study, and observe the data in great detail. Examples of the features that make this possible are: projections, zooming, date options: single frame forward, single frame backward, increase, reverse and stop. With these few options users can view different data types at different zoomed percent and at different projections for as long as they need depending on their particular interest. This can be appreciated at a low level on Figure 6 because all GIS data manipulation techniques cannot be describe textually. Only working with the system can be fully appreciated.
- Another feature of this system is high flexibility. It is completed independant of the database upon which it operates. The system retrieves all information including data format from the database. Thus, different data set can be displayed without having to change the system itself. Also, if while the client is running the system, new data is loaded into the database, the client can view the new data instantly on the screen by just clicking the mouse. This GIS is also flexible in a way that at any time users can click on a pull-down menu option, button, or icon, and a variety of operations can be performed instantly: view different data set, go back to main program, view data on a particular date, zoom in over Europe, make a query to the database, or just stop the whole system until you decide to move on again.

- The high observation capability is an additional feature of this GIS. One of the main options that allows to achieve the high observation capability are 'Rotation' increase and stop, 'Projection', and 'Zoom'. Users can rotate a data frame in any selected projection until the desired view of the geographical location under research is achieved. Besides, clients can use the 'Zoom' option to zoom in several geographical locations and display them on different windows at several zoomed percent and keep all of this plus the main user interface window on the same screen. The combination of all these options provides a great observation capability and research tool.
- Every displayed image is fully documented. A description containing the data type name, frequency, and satellite is given on the top border of the image. Additional documentation includes: a color table giving a one-to-one relationship between colors and values of the displaying data, description of color table, units values, and date for each frame.
- Easy to use user interface allows the clients to navigate through the semantic database. Users do not need to have knowledge about the database schema or the information stored on the database. By just clicking menu options, they are guided through all categories, and relations of the database, and they can retrieve all the desired semantic and spatial data at any time during the program execution.

This Geographic Information System could also be considered as an intelligent system since it is able to "remember", distinctive geographical information such as continents and clouds and then display them in other projections as point of reference for data precession, or omit them for data clarity. In addition, this system is completely independent of the data upon which it operates. Moreover, since this GIS works with a semantic DBMS, the system is efficient, and reliable.

Disadvantage

One of the main disadvantage of this system is on the displaying area. The animated movie portrayed by the system does not achieve a very high good movie speed. And this is mainly due to the use of the X-windows system to manage the display and the windows. Another minor disadvantage of this system is that it is not very portable. This is a UNIX base system, so a machine that does not have UNIX as its operating system is not able to execute the program.

Another observable problem on the system is that parallel operations cannot be performed efficiently, but this is not a system problem. This is a problem of the DBMS. As I have mention before, the semantic DBMS used by this GIS is under development at HPDRC, and up to now the system has not been implemented with this advanced feature. Once the DBMS provides that feature, no changes need to be performed to the GIS to start delivering better performance. The GIS is a client program having a semantic DBMS as the server, so it is limited by the constraints of the DBMS capability.

Data Analysis

This GIS system has also allowed us to analyze, interpreted, and study all spatial data currently stored in the database. For instance, Ocean Temperature data has been analyzed and the regions of high temperature water has been observed. We have also been able to observe how water temperature changes from one latitude to another and from one season to another. One observation has been that regions on the Atlantic Ocean close to the Tropic of Cancer has warm water during the whole summer (June-October) making conditions favorable for the development of many hurricanes and tropical storms.

Meanwhile, Ozone(TOMS) data has also been interpreted, analyzed and observed. The development of the ozone hole (regions where the total ozone column is less than

200 DU) has been monitored. For example, figure 7, 8 and 9 depict a graphical sequence of the Antarctic ozone hole formation based on monthly reading from the Nimbus-7 Satellite. In addition, figure 10 portrays daily Ozone data readings from the Meteor-3 for April 16, 1993 and figure 11 displays Ozone data for October 6, 1993. Figure 10 shows the ozone layers at the beginning of the year and as can be seen all values are greater than 230 DU. The figure 11, on the contrary, depicts Dobson Units values less than 100 DU over the South Pole (Antarctic). This values were recorded by NASA's TOMS flying on the Russia's Meteor-3 satellite on October 1993. This was the date that the ozone hole was 9 millions square miles, slightly smaller than the 1992's record ozone hole, 9.4 millions square miles. On this day, a balloon sensor flown by NOAA measured 90 DU while a spectrophotometer on the surface measured 88 DU [17]. This two maps from the same year and satellite, but different seasons show how variable ozone layers values can be from one month to the other.

Figure 7. Ozone hole starts forming

Figure 10. Ozone data at the beginning of the year

Figure 11. Day of the largest ozone hole

Ways to Improve the system

There are several procedures that can be performed to improve the GIS and achieve a better system performance. Different tools could be used to handle the data display and windows management. Tools like SRGP are good, but they are a little bit primitive. There are better tools on the market today that can provide easy to use users interface development kits and good graphic displaying techniques. This will make easy the program maintenance and will improve the animated movie tremendously.

In addition, more efficient algorithms should be developed to substitute the existing ones that run in exponential time. The computational time will be greatly reduced if linear time algorithms are developed. These minor improvements will have a visible impact in the system performance.

VIII. CONCLUSION

The design and implementation of the semantic Geographic Information System covered several areas. First, its semantic database was designed and created, and all the textual and spatial data were loaded into the database. Then, the main areas of the system: user interface, displaying process, data animation, data retrieval, and data transfer were implemented with the help of some tools and devices. Then, finally, all these components were tightly integrated to form a novel, innovative and practical Geographic Information System.

The resulting system have facilitated the interpretation, storage, manipulation, analysis and display of spatial data like: Ocean Temperature, Ozone (TOMS), and simulated SeaWiFS data. At the same time, this system have given a tremendous help in the testing process of the high performance and efficient parallel semantic DBMS being developed at HPDRC.

As I mention on section VII. - Data Analysis, with the help of the GIS system, the ocean temperature water has been analyzed, and the ozone layer has been tracking for a period of more than 14 years. When is the ocean temperature near the west coast of Africa over 30 degrees Celsius (~90 degrees F)? When the ozone hole was first detected by the TOMS instrument? or During what weather seasons the ozone layer is thinner and over what region? These are only very few of the questions that can be answer using the semantic GIS system as the main tool for all data examination, interpretation, and manipulation. This GIS guides the user to retrieve all desire information from the database and then helps them to better analyze and observe the selected spatial data using a wide variety of menu options.

In the meanwhile, the semantic DBMS has also constantly been tested. Database retrieval time has been monitored. The GIS system has been run using the ATM switch

and using the regular Ethernet, and in both cases the retrieval and transfer time was checked. Also, data stored in the database have been periodically checked to ensure that the DBMS system is been delivering protection, security and consistency for the data stored. In addition, several methods' combination obtained from the Semantic Binary Database Interface and algorithms have been implemented to retrieve data from the database and detect which one produces the optimal result. In general, test results have shown that the semantic DBMS system delivers an efficient spatial and textual data retrieval time and consistency for the data at the same time.

Future Work

Future works include the development of a Java version of the GIS system, which I have already started developing. The main purpose in implementing this system is to make the semantic GIS available through the World Wide Web.

This is client/server application that is very portable across several platforms. The client program is a Java Applet and consists of three main areas: user interface, data displaying, and data manipulation. Meanwhile, the server application handles the data retrieval (database queries) and data transfer. All these components together with the semantic database are tightly integrated to achieve the system's goals.

Once this system is completely implemented, users are going to be able to access the semantic database, view an animated movie of a particular data type, make queries, and retrieve all stored spatial and semantic information from their home through the WWW using any browser tools like Netscape or HotJava. This new system is going to offer most of the semantic GIS features including movie capabilities, zooming, rotation, single frame displaying, and many others. Also, this application is going to provide new features like: interactive database schema view that is going to allow users better navigate through the

database information, and dialog boxes that will allow clients to communicate with the database for retrieval or updates depending on the need.

APPENDICES

APPENDIX A

```
#include "sdb3.h"
#include <stdlib.h>

// create the database schema
void
CreateSchema(TDataBase *DB)
{
    do
    {
        DB->Transaction_Begin();
        NewCategory(DB, "DATATYPE");
        NewRelation(DB, "dname", "DATATYPE", "String");
        NewRelation(DB, "bytes", "DATATYPE", "Integer");
        NewRelation(DB, "cmext", "DATATYPE", "String");

        NewCategory(DB, "ColorTable");
        NewRelation(DB, "color", "ColorTable", "Binary");
        NewRelation(DB, "value", "ColorTable", "Binary");
        NewRelation(DB, "mapcolor", "ColorTable", "Binary");
        NewRelation(DB, "description", "ColorTable", "String");
        NewRelation(DB, "hastable", "DATATYPE", "ColorTable");

        NewCategory(DB, "Satellite");
        NewRelation(DB, "sname", "Satellite", "String");
        NewRelation(DB, "takeby", "DATATYPE", "Satellite");

        NewCategory(DB, "Instrument");
        NewRelation(DB, "iname", "Instrument", "String");
        NewRelation(DB, "uses", "Satellite", "Instrument");

        NewCategory(DB, "ObserProg");
        NewRelation(DB, "datadescr", "ObserProg", "String");
        NewRelation(DB, "enddate", "ObserProg", "Integer");
    }
```

```

NewRelation(DB, "startdate", "ObserProg", "Integer");
NewRelation(DB, "frequency", "ObserProg", "String");
NewRelation(DB, "progused", "Instrument", "ObserProg");

NewCategory(DB, "BITMAP");
NewRelation(DB, "realdate", "BITMAP", "Integer");
NewRelation(DB, "image", "BITMAP", "Binary");
NewRelation(DB, "date", "BITMAP", "String");
NewRelation(DB, "hasdata", "ObserProg", "BITMAP");
}
while(DB->Transaction_End() == ec_concurrency);
};

```

// check the value for ocean temperature date

long int

Check_date(long int Ndate)

```

{
    int mon_day = Ndate - 870000;
    int mon = mon_day/100;
    int nmon= mon * 100;
    int day = mon_day - nmon;
    if(day > 31)
        return ((Ndate - 31)+100);
    else
        return Ndate;
}

```

// check the index for monthly ozone files

long int

check_ind_month(long int index)

```

{
    int year = index / 100;
    int tyear = year * 100;
    int month = index - tyear;
    if(month > 12)
    {
        index += 100;
        index -= 12;
    }
    return index;
}

```

```

// check the value for Ozone date
long int
Check_ozdate(long int Odate)
{
    int year = Odate / 10000;
    int tyear= year * 10000;
    int mon_day = Odate - tyear;
    int month = mon_day / 100;
    int nmon = month * 100;
    int day = mon_day - nmon;
    if(month > 12)
    {
        Odate += 10000;
        Odate -= 1200;
    }
    return Odate;
}

// check index for daily ozone data
long int
check_index(long int ind)
{
    if(ind == 910132) ind += 69;
    else if(ind == 910229) ind += 72;
    else if(ind == 910332) ind += 69;
    else if(ind == 910431) ind += 70;
    else if(ind == 910532) ind += 69;
    else if(ind == 910631) ind += 70;
    else if(ind == 910732) ind += 69;
    else if(ind == 910832) ind += 69;
    else if(ind == 910931) ind += 70;
    else if(ind == 911032) ind += 69;
    else if(ind == 911131) ind += 70;
    else if(ind == 911232) ind += 8869;
    else if(ind == 920132) ind += 69;
    else if(ind == 920230) ind += 71;
    else if(ind == 920332) ind += 69;
    else if(ind == 920431) ind += 70;
    else if(ind == 920532) ind += 69;
    else if(ind == 920631) ind += 70;
    else if(ind == 920732) ind += 69;
    else if(ind == 920832) ind += 69;

```

```

else if(ind == 920931) ind += 70;
else if(ind == 921032) ind += 69;
else if(ind == 921131) ind += 70;
else if(ind == 921232) ind += 8869;
else if(ind == 930132) ind += 69;
else if(ind == 930229) ind += 72;
else if(ind == 930332) ind += 69;
else if(ind == 930431) ind += 70;
else if(ind == 930532) ind += 69;
else if(ind == 930631) ind += 70;
else if(ind == 930732) ind += 69;
else if(ind == 930832) ind += 69;
else if(ind == 930931) ind += 70;
else if(ind == 931032) ind += 69;
else if(ind == 931131) ind += 70;
else if(ind == 931232) ind += 8869;
else if(ind == 940132) ind += 69;
else if(ind == 940229) ind += 72;
else if(ind == 940332) ind += 69;
else if(ind == 940431) ind += 70;
else if(ind == 940532) ind += 69;
else if(ind == 940631) ind += 70;
else if(ind == 940732) ind += 69;
else if(ind == 940832) ind += 69;
else if(ind == 940931) ind += 70;
else if(ind == 941032) ind += 69;
else if(ind == 941131) ind += 70;

return ind;
}

// load the continents and the continents's outline
void
LoadCont(TDataBase *DB)
{
    char  Name[100], files[100];
    byte  buff[1024];
    OK_code Error;
    int   y, ind = 8700;
    FILE  *in_pt;
    Var   B, V;
    size_t read;

```

```

do
{ DB->Transaction_Begin();

    // loading the continents
    sprintf(Name, "%d%s", ind, "ct");
    in_pt = fopen("Conts", "rb");
    printf("loading %s\n", "Conts");

    V = DB->NewAbstract("BITMAP");
    B = DB->NewBinary();
    for (y=0; y<180; y++)
    {
        read = fread(buff, 1, 360, in_pt);
        dbwrite(buff, 1, 360, B);
    }
    V.Relate("BITMAP::date", Name);
    V.Relate("BITMAP::image", B);
    dbclose(B);
    fclose(in_pt);

    // loading the edges
    sprintf(Name, "%d%s", ind, "ed");

    in_pt = fopen("edge", "rb");
    printf("loading %s\n", "edge");

    V = DB->NewAbstract("BITMAP");
    B = DB->NewBinary();
    for (y=0; y<180; y++)
    {
        read = fread(buff, 1, 360, in_pt);
        dbwrite(buff, 1, 360, B);
    }
    V.Relate("BITMAP::date", Name);
    V.Relate("BITMAP::image", B);
    dbclose(B);
    fclose(in_pt);

    Error = DB->Transaction_End();
}
while (Error == ec_concurrency);
}

```

```

// load the binary files for each of the data sets
void
LoadImage(TDataBase *DB)
{
    char  Name[100], files[100];
    byte  buff[1024];
    long int Rdate = 870101, OzDate = 781101, Seadate = 950301;
    long int Mtdate = 910801;
    OK_code Error;
    int  y, ind;

    // load ocean temperature data files
    do
    { DB->Transaction_Begin();

        for(ind = 8701; ind < 8753; ind++)
        {
            sprintf(files, "%s%d", "/home/hpdrc-d3/data/mariosplace/oceantemp/Temp-", ind);
            sprintf(Name, "%s%d", "ot", ind);

            FILE* in_pt = fopen(files, "rb");
            printf("loading %s\n", files);

            Var V = DB->NewAbstract("BITMAP");
            Var B = DB->NewBinary();
            size_t read;
            for (y=0; y < 180; y++)
            {
                read = fread(buff, 1, 360, in_pt);
                dbwrite(buff, 1, 360, B);
            }

            V.Relate("BITMAP::date", Name);
            V.Relate("BITMAP::image", B);
            V.Relate("BITMAP::realdate", Rdate);
            Rdate += 7;
            Rdate = Check_date(Rdate);

            dbclose(B);
            fclose(in_pt);
        };
    }
    while(DB->Transaction_End() == ec_concurrency);
}

```



```

printf("all ocean temp data was loaded \n");

// load meteor-3 monthly ozone data files
do
{
    DB->Transaction_Begin();
    for(ind = 9108; ind < 9413; ind++)
    {
        ind = check_ind_month(ind);
        sprintf(files, "%s%d.%s",
                "/home/hpdrc-d3/data/mariosplace/meteor3/monthly/m3_", ind, "mar");
        sprintf(Name, "%s%li", "m3", ind);

        FILE* in_pt = fopen(files, "rb");
        printf("loading %s\n", files);

        Var V = DB->NewAbstract("BITMAP");
        Var B = DB->NewBinary();
        size_t read;
        for (y=0; y < 180; y++)
        {
            read = fread(buff, 1, 576, in_pt);
            dbwrite(buff, 1, 576, B);
        }
        V.Relate("BITMAP::date", Name);
        V.Relate("BITMAP::image", B);
        V.Relate("BITMAP::realdate", Mtdate);
        Mtdate += 100;
        Mtdate = Check_ozdate(Mtdate);

        dbclose(B);
        fclose(in_pt);
    };
}
while(DB->Transaction_End() == ec_concurrency);
printf("all meteor3 monthly dates were loaded\n");

// load meteor-3 daily ozone data files
do
{
    DB->Transaction_Begin();
    for(ind = 910822; ind < 941228; ind++)
    {
        ind = check_index(ind);
    }
}

```

```

    sprintf(files, "%s%d.%s",
              "/home/hpdrc-d3/data/mariosplace/meteor3/daily/ga", ind, "mar");
    sprintf(Name, "%li", ind);

    FILE* in_pt = fopen(files, "rb");
    printf("loading %s\n", files);

    Var V = DB->NewAbstract("BITMAP");
    Var B = DB->NewBinary();
    size_t read;
    for (y=0; y < 180; y++)
    {
        read = fread(buff, 1, 576, in_pt);
        dbwrite(buff, 1, 576, B);
    }
    V.Relate("BITMAP::date", Name);
    V.Relate("BITMAP::image", B);
    V.Relate("BITMAP::realdate", ind);

    dbclose(B);
    fclose(in_pt);
};
}
while(DB->Transaction_End() == ec_concurrency);
printf("all meteor3 daily data was loaded\n");

// load nimbus-7 monthly ozone data files
do
{
    DB->Transaction_Begin();
    for(ind = 7811; ind < 9305; ind++)
    {
        ind = check_ind_month(ind);
        sprintf(files, "%s%d.%s",
                  "/home/hpdrc-d3/data/mariosplace/nimbus7/monthly/", ind, "mar");
        sprintf(Name, "%s%li", "m", ind);

        FILE* in_pt = fopen(files, "rb");
        printf("loading %s\n", files);

        Var V = DB->NewAbstract("BITMAP");
        Var B = DB->NewBinary();
        size_t read;
        for (y=0; y < 180; y++)

```

```

    {
        read = fread(buff, 1, 576, in_pt);
        dbwrite(buff, 1, 576, B);
    }
    V.Relate("BITMAP::date", Name);
    V.Relate("BITMAP::image", B);
    V.Relate("BITMAP::realdate", OzDate);
    OzDate += 100;
    OzDate = Check_ozdate(OzDate);

    dbclose(B);
    fclose(in_pt);
}
}
while(DB->Transaction_End() == ec_concurrency);
printf("all nimbus7 monthly data was loaded\n");

//load nimbus-7 daily ozone data files
do
{
    DB->Transaction_Begin();
    for(ind = 910101; ind < 930506; ind++)
    {
        ind = check_index(ind);
        sprintf(files, "%s%d.%s",
                "/home/hpdrc-d3/data/mariosplace/nimbus7/daily/d", ind, "mar");
        sprintf(Name, "%s%li", "d", ind);

        FILE* in_pt = fopen(files, "rb");
        printf("loading %s\n", files);

        Var V = DB->NewAbstract("BITMAP");
        Var B = DB->NewBinary();
        size_t read;
        for (y=0; y < 180; y++)
        {
            read = fread(buff, 1, 576, in_pt);
            dbwrite(buff, 1, 576, B);
        }
        V.Relate("BITMAP::date", Name);
        V.Relate("BITMAP::image", B);
        V.Relate("BITMAP::realdate", ind);
    }
}

```

```

        dbclose(B);
        fclose(in_pt);
    }
}
while(DB->Transaction_End() == ec_concurrency);
printf("all nimbus7 daily data was loaded\n");

//load SeaWiFS data files
do
{
    DB->Transaction_Begin();
    for(ind = 0; ind < 8; ind++)
    {
        sprintf(files, "%s%d",
                "/home/hpdrc-d3/data/mariosplace/seawifs/original/seamap", ind);
        sprintf(Name, "%s%d", "seamap", ind);

        FILE* in_pt = fopen(files, "rb");
        printf("loading %s\n", files);

        Var V = DB->NewAbstract("BITMAP");
        Var B = DB->NewBinary();
        size_t read;
        for(y=0; y < 180; y++)
        {
            read = fread(buff, 1, 360, in_pt);
            dbwrite(buff, 1, 360, B);
        }
        V.Relate("BITMAP::date", Name);
        V.Relate("BITMAP::image", B);
        V.Relate("BITMAP::realdate", Seadate);

        dbclose(B);
        fclose(in_pt);
    };
    Error = DB->Transaction_End();
}
while (Error == ec_concurrency);
printf("all seawifs data was loaded\n");
}

```

```

// load the textual information and add the relation
void
LoadData(TDataBase* DB)
{
    OK_code Error;
    char datekey[12];
    short int color_oz[652], color_ot[256], color_sea[256];
    long int ind;
    short int col, i=0, temp, x;
    short int temp_numb[40], color_numb[40];
    short int ozone_val[40], ozone_col[40];
    Var Image, C, V, CM, CT[3];

    //calculate color table values for ocean temperature
    for(temp = -2; temp < 38; temp++)
    {
        col = temp;
        if(col <= 0)
            col = 43 + col;
        col += 2;
        temp_numb[i] = temp;
        color_numb[i++] = col;
    }
    col = 54;

    // calculate color table values for ozone
    for(temp = 113, i=0; temp < 583; temp += 12, i++)
    {
        ozone_col[i] = col;
        ozone_val[i] = temp;
        col += 3;
    }

    // compute color_map values for ozone
    color_oz[0] = 42;
    for(i= 1; i<101; i++)
        color_oz[i] = 52;
    for(i=101; i<600; i++)
        color_oz[i] = ((i/2)-48)/2 +51;
    for(i=600; i<650; i++)
        color_oz[i] = 176;
    color_oz[650] = 1;
}

```

```

// compute color_map values for ocean temperature
for (i=0; i<256; i++)
{
    x = 0.15*i-3;
    if (x<=0) x = 43+x;
    color_ot[i] = x+2;
}
// compute color_map values for SeaWiFS
for(i=0; i<256; i++)
    color_sea[i] = i;
color_sea[0] = 1;

do { DB->Transaction_Begin();
    // load color table info for ocean temperature
    CT[0] = DB->NewAbstract("ColorTable");
    C = DB->NewBinary();
    V = DB->NewBinary();
    CM = DB->NewBinary();
    dbwrite(color_numb, 2, 40, C);
    dbwrite(temp_numb, 2, 40, V);
    dbwrite(color_ot, 2, 256, CM);
    CT[0].Relate("ColorTable::color", C);
    CT[0].Relate("ColorTable::value", V);
    CT[0].Relate("ColorTable::mapcolor", CM);
    CT[0].Relate("ColorTable::description", "Temperature in degrees C");
    dbclose(C); dbclose(V); dbclose(CM);

    // load color table info for ozone
    CT[1] = DB->NewAbstract("ColorTable");
    C = DB->NewBinary();
    V = DB->NewBinary();
    CM = DB->NewBinary();
    dbwrite(ozone_col, 2, 40, C);
    dbwrite(ozone_val, 2, 40, V);
    dbwrite(color_oz, 2, 650, CM);
    CT[1].Relate("ColorTable::color", C);
    CT[1].Relate("ColorTable::value", V);
    CT[1].Relate("ColorTable::mapcolor", CM);
    CT[1].Relate("ColorTable::description", "Ozone Layer in Dobson Units");
    dbclose(C); dbclose(V); dbclose(CM);

    // load color table info for SeaWiFS
    CT[2] = DB->NewAbstract("ColorTable");

```

```

CM = DB->NewBinary();
dbwrite(color_sea, 2, 256, CM);
CT[2].Relate("ColorTable::description", "none");
CT[2].Relate("ColorTable::mapcolor", CM);
dbclose(CM);

// load observation program data for Ocean Temperature
Var Obser[13];
Obser[0] = DB->NewAbstract("ObserProg");
Obser[0].Relate("ObserProg::datadescr", "Weekly Ocean Temperature Data");
Obser[0].Relate("ObserProg::frequency", "Weekly");
Obser[0].Relate("ObserProg::startdate", (long)870101);
Obser[0].Relate("ObserProg::enddate", (long)871217);
for(ind = 8701; ind < 8753; ind++)
{
    sprintf(datekey, "%s%li", "ot", ind);
    Image = RangeQuery(TheRelation("BITMAP::date", DB), datekey);
    Obser[0].Relate("ObserProg::hasdata", Image);
}
// load observation program data for ozone
Obser[1] = DB->NewAbstract("ObserProg");
Obser[1].Relate("ObserProg::datadescr", "Nimbus-7 Monthly Ozone Data");
Obser[1].Relate("ObserProg::frequency", "Monthly");
Obser[1].Relate("ObserProg::startdate", (long)781101);
Obser[1].Relate("ObserProg::enddate", (long)930401);
for(ind = 7811; ind < 9305; ind++)
{
    ind = check_ind_month(ind);
    sprintf(datekey, "%s%li", "m", ind);
    Image = RangeQuery(TheRelation("BITMAP::date", DB), datekey);
    Obser[1].Relate("ObserProg::hasdata", Image);
}
Obser[2] = DB->NewAbstract("ObserProg");
Obser[2].Relate("ObserProg::datadescr", "Nimbus-7 Daily Ozone Data");
Obser[2].Relate("ObserProg::frequency", "Daily");
Obser[2].Relate("ObserProg::startdate", (long)910101);
Obser[2].Relate("ObserProg::enddate", (long)930505);
for(ind = 910101; ind < 930506; ind++)
{
    ind = check_index(ind);
    sprintf(datekey, "%s%li", "d", ind);
    Image = RangeQuery(TheRelation("BITMAP::date", DB), datekey);
    Obser[2].Relate("ObserProg::hasdata", Image);
}

```



```

}
Obser[3] = DB->NewAbstract("ObserProg");
Obser[3].Relate("ObserProg::datadescr", "Meteor-3 Monthly Ozone Data");
Obser[3].Relate("ObserProg::frequency", "Monthly");
Obser[3].Relate("ObserProg::startdate", (long)910801);
Obser[3].Relate("ObserProg::enddate", (long)941201);
for(ind = 9108; ind < 9413; ind++)
{
    ind = check_ind_month(ind);
    sprintf(datekey, "%s%li", "m3", ind);
    Image = RangeQuery(TheRelation("BITMAP::date", DB), datekey);
    Obser[3].Relate("ObserProg::hasdata", Image);
}
Obser[4] = DB->NewAbstract("ObserProg");
Obser[4].Relate("ObserProg::datadescr", "Meteor-3 Daily Ozone Data");
Obser[4].Relate("ObserProg::frequency", "Daily");
Obser[4].Relate("ObserProg::startdate", (long)910822);
Obser[4].Relate("ObserProg::enddate", (long)941227);
for(ind = 910822; ind < 941228; ind++)
{
    ind = check_index(ind);
    sprintf(datekey, "%li", ind);
    Image = RangeQuery(TheRelation("BITMAP::date", DB), datekey);
    Obser[4].Relate("ObserProg::hasdata", Image);
}
// load observation program data for SeaWiFS
Obser[5] = DB->NewAbstract("ObserProg");
Obser[5].Relate("ObserProg::datadescr", "SeaStar Daily SeaWiFS Data");
Obser[5].Relate("ObserProg::frequency", "Daily");
Obser[5].Relate("ObserProg::startdate", (long)940301);
Obser[5].Relate("ObserProg::enddate", (long)940301);
Image = RangeQuery(TheRelation("BITMAP::date", DB), "seamap0");
Obser[5].Relate("ObserProg::hasdata", Image);

Obser[6] = DB->NewAbstract("ObserProg");
Obser[6].Relate("ObserProg::datadescr", "SeaStar Daily SeaWiFS Data");
Obser[6].Relate("ObserProg::frequency", "Daily");
Obser[6].Relate("ObserProg::startdate", (long)940301);
Obser[6].Relate("ObserProg::enddate", (long)940301);
Image = RangeQuery(TheRelation("BITMAP::date", DB), "seamap1");
Obser[6].Relate("ObserProg::hasdata", Image);

Obser[7] = DB->NewAbstract("ObserProg");

```

```

Obser[7].Relate("ObserProg::datadescr", "SeaStar Daily SeaWiFS Data");
Obser[7].Relate("ObserProg::frequency", "Daily");
Obser[7].Relate("ObserProg::startdate", (long)940301);
Obser[7].Relate("ObserProg::enddate", (long)940301);
Image = RangeQuery(TheRelation("BITMAP::date", DB), "seamap2");
Obser[7].Relate("ObserProg::hasdata", Image);

```

```

Obser[8] = DB->NewAbstract("ObserProg");
Obser[8].Relate("ObserProg::datadescr", "SeaStar Daily SeaWiFS Data");
Obser[8].Relate("ObserProg::frequency", "Daily");
Obser[8].Relate("ObserProg::startdate", (long)940301);
Obser[8].Relate("ObserProg::enddate", (long)940301);
Image = RangeQuery(TheRelation("BITMAP::date", DB), "seamap3");
Obser[8].Relate("ObserProg::hasdata", Image);

```

```

Obser[9] = DB->NewAbstract("ObserProg");
Obser[9].Relate("ObserProg::datadescr", "SeaStar Daily SeaWiFS Data");
Obser[9].Relate("ObserProg::frequency", "Daily");
Obser[9].Relate("ObserProg::startdate", (long)940301);
Obser[9].Relate("ObserProg::enddate", (long)940301);
Image = RangeQuery(TheRelation("BITMAP::date", DB), "seamap4");
Obser[9].Relate("ObserProg::hasdata", Image);

```

```

Obser[10] = DB->NewAbstract("ObserProg");
Obser[10].Relate("ObserProg::datadescr", "SeaStar Daily SeaWiFS Data");
Obser[10].Relate("ObserProg::frequency", "Daily");
Obser[10].Relate("ObserProg::startdate", (long)940301);
Obser[10].Relate("ObserProg::enddate", (long)940301);
Image = RangeQuery(TheRelation("BITMAP::date", DB), "seamap5");
Obser[10].Relate("ObserProg::hasdata", Image);

```

```

Obser[11] = DB->NewAbstract("ObserProg");
Obser[11].Relate("ObserProg::datadescr", "SeaStar Daily SeaWiFS Data");
Obser[11].Relate("ObserProg::frequency", "Daily");
Obser[11].Relate("ObserProg::startdate", (long)940301);
Obser[11].Relate("ObserProg::enddate", (long)940301);
Image = RangeQuery(TheRelation("BITMAP::date", DB), "seamap6");
Obser[11].Relate("ObserProg::hasdata", Image);

```

```

Obser[12] = DB->NewAbstract("ObserProg");
Obser[12].Relate("ObserProg::datadescr", "SeaStar Daily SeaWiFS Data");
Obser[12].Relate("ObserProg::frequency", "Daily");
Obser[12].Relate("ObserProg::startdate", (long)940301);

```

```

Obser[12].Relate("ObserProg::enddate", (long)940301);
Image = RangeQuery(TheRelation("BITMAP::date", DB), "seamap7");
Obser[12].Relate("ObserProg::hasdata", Image);

```

```

// load Instrument data

```

```

Var Inst[11];
Inst[0] = DB->NewAbstract("Instrument");
Inst[0].Relate("Instrument::iname", "Temp Scanner");
Inst[0].Relate("Instrument::progused", Obser[0]);
Inst[1] = DB->NewAbstract("Instrument");
Inst[1].Relate("Instrument::iname", "N-7 TOMS");
Inst[1].Relate("Instrument::progused", Obser[1]);
Inst[1].Relate("Instrument::progused", Obser[2]);
Inst[2] = DB->NewAbstract("Instrument");
Inst[2].Relate("Instrument::iname", "M-3 TOMS");
Inst[2].Relate("Instrument::progused", Obser[3]);
Inst[2].Relate("Instrument::progused", Obser[4]);
Inst[3] = DB->NewAbstract("Instrument");
Inst[3].Relate("Instrument::iname", "Sensor 1");
Inst[3].Relate("Instrument::progused", Obser[5]);
Inst[4] = DB->NewAbstract("Instrument");
Inst[4].Relate("Instrument::iname", "Sensor 2");
Inst[4].Relate("Instrument::progused", Obser[6]);
Inst[5] = DB->NewAbstract("Instrument");
Inst[5].Relate("Instrument::iname", "Sensor 3");
Inst[5].Relate("Instrument::progused", Obser[7]);
Inst[6] = DB->NewAbstract("Instrument");
Inst[6].Relate("Instrument::iname", "Sensor 4");
Inst[6].Relate("Instrument::progused", Obser[8]);
Inst[7] = DB->NewAbstract("Instrument");
Inst[7].Relate("Instrument::iname", "Sensor 5");
Inst[7].Relate("Instrument::progused", Obser[9]);
Inst[8] = DB->NewAbstract("Instrument");
Inst[8].Relate("Instrument::iname", "Sensor 6");
Inst[8].Relate("Instrument::progused", Obser[10]);
Inst[9] = DB->NewAbstract("Instrument");
Inst[9].Relate("Instrument::iname", "Sensor 7");
Inst[9].Relate("Instrument::progused", Obser[11]);
Inst[10] = DB->NewAbstract("Instrument");
Inst[10].Relate("Instrument::iname", "Sensor 8");
Inst[10].Relate("Instrument::progused", Obser[12]);

```

```

// load satellite data

```

```

Var Satel[4];
Satel[0] = DB->NewAbstract("Satellite");
Satel[0].Relate("Satellite::sname", "none");
Satel[0].Relate("Satellite::uses", Inst[0]);

Satel[1] = DB->NewAbstract("Satellite");
Satel[1].Relate("Satellite::sname", "Nimbus-7");
Satel[1].Relate("Satellite::uses", Inst[1]);

Satel[2] = DB->NewAbstract("Satellite");
Satel[2].Relate("Satellite::sname", "Meteor-3");
Satel[2].Relate("Satellite::uses", Inst[2]);

Satel[3] = DB->NewAbstract("Satellite");
Satel[3].Relate("Satellite::sname", "SeaStar");
Satel[3].Relate("Satellite::uses", Inst[3]);
Satel[3].Relate("Satellite::uses", Inst[4]);
Satel[3].Relate("Satellite::uses", Inst[5]);
Satel[3].Relate("Satellite::uses", Inst[6]);
Satel[3].Relate("Satellite::uses", Inst[7]);
Satel[3].Relate("Satellite::uses", Inst[8]);
Satel[3].Relate("Satellite::uses", Inst[9]);
Satel[3].Relate("Satellite::uses", Inst[10]);

// load datatype data
Var D = DB->NewAbstract("DATATYPE");
D.Relate("DATATYPE::dname", "Ocean Temp");
D.Relate("DATATYPE::bytes", (long)360);
D.Relate("DATATYPE::cmext", "ct");
D.Relate("DATATYPE::hastable", CT[0]);
D.Relate("DATATYPE::takeby", Satel[0]);

D = DB->NewAbstract("DATATYPE");
D.Relate("DATATYPE::dname", "Ozone");
D.Relate("DATATYPE::bytes", (long)576);
D.Relate("DATATYPE::cmext", "ed");
D.Relate("DATATYPE::hastable", CT[1]);
D.Relate("DATATYPE::takeby", Satel[1]);
D.Relate("DATATYPE::takeby", Satel[2]);

D = DB->NewAbstract("DATATYPE");
D.Relate("DATATYPE::dname", "SeaWiFS");
D.Relate("DATATYPE::bytes", (long)360);

```

```

        D.Relate("DATATYPE::cmext", "ed");
        D.Relate("DATATYPE::hastable", CT[2]);
        D.Relate("DATATYPE::takeby", Satel[3]);

    }
    while(DB->Transaction_End() == ec_concurrency);
};

int
main(void)
{
    TDataBase* DB;

    DB = OpenDataBase("/home/hpdrc-d2/data/Earth.DB");
    CreateSchema(DB);    printf("CreateSchema done!\n");
    LoadImage(DB);      printf("LoadImage done!\n");
    LoadCont(DB);       printf("LoadCont done!\n");
    LoadData(DB);       printf("LoadData done!\n");
    CloseDataBase(DB);
    return 0;
};

```

APPENDIX B

```
#define X11
#include "sdb3.h"
#include "srgplocal.h"
#include "color1.h"
#include "sort.h"
#include "math.h"
#include "time.h"

#define d_dpy srgpx__display
#define d_act srgp__curActiveCanvasSpec
#define locator srgp__locator_measure

XImage      *xi;
TDataBase *DB;
char        dnames[4][12], satelname[4][10], instrs[10][10], freqs[4][10];
int         menu=0, area=0, indrd, dindex=0, sindex=0, iindex=0, findex=0;
int         f, fn, Pw, Ph, Dw, Dh, Px, Py, Ct=0, Rot=0;
short int   color[651];
int         P=5, D=0, R=2, Dt=5, Cl=2, Sf=0, Zo=0, Zp=6, Zcheck=0;
int         g0[180], pcol[64800], g1[180], hpw, TorO=0;
long int    itp, iotp;
time_t      *tp;
byte        FileBuff[200000];
unsigned    char cmap[64800];
size_t      Fptr;
char        *map[64800], *key[32400];
float       ccos[360], ssin[360], sqt;
int         freq=0, satellite=0, satel_area=0, freq_area=0, inst_area=0;
int         limit, vd = 1, opt = 7, flag=1, Zoom=0, See_date=0, Dc=0, inst=0;
int         max_ind, initxl, initxr, inityl, inityr, cloud=0;
int         Pwt=360, Pht=180, fv=2, Dwt=720, Dht=360, prev_fv;

int
fastread(void* b,size_t nr,size_t size)
{
    memcpy(b, FileBuff + Fptr, nr); Fptr += nr;
}
```

```

// send zoomed frame to the screen
void
send_zoom_frame()
{
    XPutImage(d_dpy,d_act.drawable.xid,d_act.gc_frame,xi,0,0,4,4,Dw,Dh);
}

// send regular frame to the screen
void
send_frame()
{
    XPutImage(d_dpy,d_act.drawable.xid,d_act.gc_frame,xi,0,0,hpw,220,Dw,Dh);
}

// set the input mode for the program devices
void
SetInputMode()
{
    SRGP_setInputMode(KEYBOARD, EVENT);
    SRGP_setKeyboardProcessingMode(RAW);
    SRGP_setInputMode (LOCATOR, EVENT);
    SRGP_setLocatorButtonMask(LEFT_BUTTON_MASK);
}

// retrieve color table and display it on the screen
void
Color_Chart(char *dataname)
{
    char str[10], data1[15], data2[30], data3[14], data4[15];
    int low_y = 528, upper_y = 540;
    short int colort[40], valuet[40];
    int i, d3;

    Var OT = RangeQuery(TheRelation("DATATYPE::dname", DB), dataname);
    Var table = OT["DATATYPE::hastable"];
    Var desc = table["ColorTable::description"];
    if(strcmp(pChar(desc), "none") != 0)
    {
        Var colors=table["ColorTable::color"];
        Var values=table["ColorTable::value"];
        Fptr=0;
        dbread(FileBuff, 1, sizeof(FileBuff), colors);
        fastread(colort, 80, 1); dbclose(colors);
    }
}

```



```

    Fptr=0;
    dbread(FileBuff, 1, sizeof(FileBuff), values);
    fastread(valuet, 80, 1); dbcloses(values);
    SRGP_setColor(0);
    SRGP_fillRectangleCoord(740, 555, 858, 590);
    SRGP_setColor(1);
    strncpy(data1, pChar(desc), 14);
    for(i=0; i<15; i++) data4[i]= data1[i];
    data4[i]='\0';
    SRGP_text(SRGP_defPoint(745,580), data1);
    strncpy(data2, pChar(desc), 29);
    for(i=15, d3=0; i<29; i++, d3++) data3[d3] = data2[i];
    data3[d3]='\0';
    SRGP_text(SRGP_defPoint(753,560), data3);

    for(i = 0; i<40; i++)
    {
        SRGP_setColor(colort[i]);
        SRGP_fillRectangleCoord(780, low_y, 792, upper_y);
        sprintf(str, "%2d", valuet[i]);
        SRGP_setColor(1);
        SRGP_text(SRGP_defPoint(798, upper_y-10), str);
        low_y -= 12; upper_y -= 12;
    }
}

// set top bar menu on the main window
void
Set_Menu()
{
    int xl=493, xdl=290, ydl=20;

    SRGP_setColor(47);
    SRGP_fillRectangleCoord(0, 608, 860, 660);
    SRGP_setColor(48);
    SRGP_fillRectangleCoord(0, 610, 860, 645);
    SRGP_setColor(1);
    SRGP_setLineWidth(3);
    SRGP_lineCoord(0, 609, 860, 609);
    SRGP_lineCoord(0, 643, 860, 643);

```

```

SRGP_setColor(49);
SRGP_text(SRGP_defPoint(40,620), "Projection");
SRGP_text(SRGP_defPoint(40,618), " ");
SRGP_text(SRGP_defPoint(160,620), "Data");
SRGP_text(SRGP_defPoint(160,618), " ");
SRGP_text(SRGP_defPoint(280,620), "Rotation");
SRGP_text(SRGP_defPoint(280,618), " ");
SRGP_text(SRGP_defPoint(400,620), "Date");
SRGP_text(SRGP_defPoint(416,618), " ");
SRGP_text(SRGP_defPoint(760,620), "Exit");
SRGP_text(SRGP_defPoint(760,618), " ");
SRGP_text(SRGP_defPoint(520,620), "Cloud");
SRGP_text(SRGP_defPoint(520,618), " ");
SRGP_text(SRGP_defPoint(640,620), "Zoom");
SRGP_text(SRGP_defPoint(640,618), " ");

SRGP_setLineWidth(1);
SRGP_setColor(1);
SRGP_rectangleCoord(xdl-1, ydl-1, xdl+201, ydl+22);
SRGP_setColor(48);
SRGP_fillRectangleCoord(xdl+2, ydl+2, xdl+198, ydl+18);
SRGP_setLineWidth(2); SRGP_setColor(0);
SRGP_lineCoord(xdl+1, ydl+19, xdl+199, ydl+19);
SRGP_lineCoord(xdl+1, ydl+1, xdl+1, ydl+19);
SRGP_setColor(50);
SRGP_lineCoord(xdl+1, ydl+1, xdl+199, ydl+1);
SRGP_lineCoord(xdl+199, ydl+1, xdl+199, ydl+19);

SRGP_setLineWidth(1);
SRGP_setColor(1);
SRGP_rectangleCoord(xl-1, ydl-1, xl+20, ydl+22);
SRGP_setColor(48);
SRGP_fillRectangleCoord(xl+2, ydl+2, xl+17, ydl+18);
SRGP_setLineWidth(2); SRGP_setColor(0);
SRGP_lineCoord(xl+1, ydl+19, xl+18, ydl+19);
SRGP_lineCoord(xl+1, ydl+1, xl+1, ydl+19);
SRGP_setColor(50);
SRGP_lineCoord(xl+1, ydl+1, xl+18, ydl+1);
SRGP_lineCoord(xl+18, ydl+1, xl+18, ydl+19);
SRGP_setColor(1);
SRGP_text(SRGP_defPoint(495, 26), "->");

```

```

xl = 267;
SRGP_setLineWidth(1);
SRGP_setColor(1);
SRGP_rectangleCoord(xl-1, ydl-1, xl+20, ydl+22);
SRGP_setColor(48);
SRGP_fillRectangleCoord(xl+2, ydl+2, xl+17, ydl+18);
SRGP_setLineWidth(2); SRGP_setColor(0);
SRGP_lineCoord(xl+1, ydl+19, xl+18, ydl+19);
SRGP_lineCoord(xl+1, ydl+1, xl+1, ydl+19);
SRGP_setColor(50);
SRGP_lineCoord(xl+1, ydl+1, xl+18, ydl+1);
SRGP_lineCoord(xl+18, ydl+1, xl+18, ydl+19);
SRGP_setColor(1);
SRGP_text(SRGP_defPoint(269, 26), "<-");
}

// set the format for the circular (Orthographic and Stereographic) projections
void
Set_Format_O()
{
    int xl = 210, yl = 80, xr = 570, yr = 440;

    SRGP_setColor(0);
    SRGP_fillRectangleCoord(28, 75, 753, 444);
    SRGP_setColor(1);
    SRGP_fillEllipse(SRGP_defRectangle(205, 75, 575, 445));
    SRGP_setColor(2);
    SRGP_fillEllipse(SRGP_defRectangle(211, 81, 569, 439));
    SRGP_setColor(1);
    SRGP_setLineWidth(2);
    SRGP_lineCoord(xl+180, yl, xr-180, yr);
    SRGP_lineCoord(xl, yl+180, xr, yr-180);
    SRGP_ellipseArc(SRGP_defRectangle(xl+60, yl, xr-60, yr), 90.0, 270.0);
    SRGP_ellipseArc(SRGP_defRectangle(xl+60, yl, xr-60, yr), 270.0, 90.0);
    SRGP_ellipseArc(SRGP_defRectangle(xl+120, yl, xr-120, yr), 90.0, 270.0);
    SRGP_ellipseArc(SRGP_defRectangle(xl+120, yl, xr-120, yr), 270.0, 90.0);
    SRGP_lineCoord(xl+40, yr-60, xr-40, yr-60);
    SRGP_lineCoord(xl+40, yl+60, xr-40, yl+60);
    SRGP_lineCoord(xl+10, yl+120, xr-10, yl+120);
    SRGP_lineCoord(xl+10, yr-120, xr-10, yr-120);
}

```

```

// make mathematical computation for the different projections
void
Startup(int bts, const char exts[])
{
    int i, j, k, x, y, cx, cy;
    char *ff;
    float R;

    cloud = 0;
    fn = 0;
    for (i=0; i<360; i++) ccos[i] = cos(3.14159265*i/180);
    for (i=0; i<360; i++) ssin[i] = sin(3.14159265*i/180);

    switch(opt)
    {
        case 0 : //Mercator
            Pw = Pwt; Ph = Pht; f=fv; Dw=Dwt; Dh=Dht; hpw=30;
            xi = XGetImage(d_dpy,d_act.drawable.xid,hpw,220,Dw,Dh,
                          AllPlanes, ZPixmap);
            if(bts == 576 || exts[0] == 'c')
            {
                for(y=0; y<Ph; y++)
                {
                    for(x=0; x<Pw; x++)
                    {
                        cx = x;
                        if(y != 0)
                            cy = 90-(ccos[y]/ssin[y])*180.0/3.14159265;
                        else
                            cy = 0;
                        if(cy >= Ph) cy = Ph - 1;
                        if(cy < 0) cy = 0;
                        if((f*(cx+cy*Dw)) < (Dw*Dh))
                            map[x+Pw*y] = &xi->data[f*(cx+cy*Dw)];
                    }
                }
                for (y=0; y<Ph; y++)
                {
                    g1[y] = f;
                    if(y < Ph-1)
                        g1[y] = (map[Pw*y+Pw] - map[Pw*y])/Dw;
                    if(g1[y] < 0) g1[y] = -g1[y];
                    if(g1[y] == 0) g1[y] = f;
                }
            }
        }
    }

```

```

    }
}
else
{
    for(y=0; y<Ph; y++)
    {
        for(x=0; x<Pw; x++)
        {
            cx = x;
            R = (ccos[y]/ssin[y])*180.0/3.14159265;
            cy = 90 + R;
            if (cy >= Ph) cy = Ph-1;
            if (cy < 0) cy = 0;
            map[x + Pw*(Ph-1-y)] = &xi->data[f*(cx + cy*Dw)];
        }
    }
    for(y=0; y<Ph; y++)
    {
        g0[y] = 1;
        if ( y < Ph-1 )
            g0[y] = (map[Pw*y+Pw]-map[Pw*y])/(f*Dw);
        if ( g0[y] == 0 ) g0[y] = 1;
    }
}
break;

```

case 3 : // Mollweid Homolographic

```

Pw = Pwt; Ph = Pht; f=fv; Dw=Dwt; Dh=Dht; hpw=30;
xi = XGetImage(d_dpy, d_act.drawable.xid, hpw, 220, Dw, Dh,
               AllPlanes, ZPixmap);
for(y=0; y<Ph; y++)
{
    for(x=0; x<Pw; x++)
    {
        cx = Ph+(x-Ph)*sqrt(1.0-((y-90.0)*(y-90.0))/8100.0);
        cy = y;
        if((f*(cx+cy*Dw)) < (Dw*Dh))
            map[x+Pw*y] = &xi->data[f*(cx+cy*Dw)];
    }
}
for(y=0; y<Ph; y++) g0[y] = 1;
break;

```

```

case 4 : // Stereographic
    Pw = 180; Ph=180; f=4; R=45; Dw=360; Dh=360; hpw = 210;
    xi = XGetImage(d_dpy, d_act.drawable.xid, hpw, 220, Dw, Dh,
                  AllPlanes, ZPixmap);
    for(y=0; y<Ph; y++)
    {
        for(x=0; x<Pw; x++)
        {
            cx = R*(1-ssin[y]*ccos[x]/(1+ssin[y]*ssin[x]));
            cy = R*(1-ccos[y]/(1+ssin[y]*ssin[x]));
            if((f*(cx+cy*Dw)) < (Dw*Dh))
                map[x+Pw*y] = &xi->data[f*(cx+cy*Dw)];
        }
    }
    break;

case 6 : // Sinusoidal
    Pw = Pwt; Ph=Pht; f=fv; Dw=Dwt; Dh=Dht; hpw=30;
    xi = XGetImage(d_dpy, d_act.drawable.xid, hpw, 220, Dw, Dh,
                  AllPlanes, ZPixmap);
    for(y=0; y<Ph; y++)
    {
        for(x=0; x<Pw; x++)
        {
            cx = Ph+ssin[y]*(x-Ph);
            cy = y;
            if((f*(cx+cy*Dw)) < (Dw*Dh))
                map[x+Pw*y] = &xi->data[f*(cx+cy*Dw)];
        }
    }
    for(y=0; y<Ph; y++) g0[y] = 1;
    break;

case 7 : // Orthogonal
    Pw = Pwt; Ph = Pht; f=fv; Dw=Dwt; Dh=Dht; hpw=30;
    xi = XGetImage(d_dpy, d_act.drawable.xid, hpw, 220, Dw, Dh,
                  AllPlanes, ZPixmap);
    for(y=0; y<Ph; y++)
    {
        for(x=0; x<Pw; x++)
        {
            cx = x;
            cy = y;

```

```

        if((f*(cx+cy*Dw)) < (Dw*Dh))
            map[x+Pw*y] = &xi->data[f*(cx+cy*Dw)];
        g0[y] = 1;
    }
}
break;

```

case 8 : // Orthographic

```

Pw = 180; hpw=210; Ph=180; f=4; R=45; Dw=360; Dh=360;
xi = XGetImage(d_dpy, d_act.drawable.xid, hpw, 220, Dw, Dh,
               AllPlanes, ZPixmap);
for(y=0; y<Ph; y++)
{
    for(x=0; x<Pw; x++)
    {
        cx = R*(1-ccos[x]*ssin[y]);
        cy = R*(1-ccos[y]);
        if((f*(cx+cy*Dw)) < (Dw*Dh))
            map[x+Pw*y] = &xi->data[f*(cx+cy*Dw)];
        g0[y] = 1;
    }
}
break;

```

default :

```
break;
```

```
}
```

```
for(y=1; y<Ph-1; y++)
```

```

{
    for(x=1; x<Pw-1; x++)
    {
        if (map[x+Pw*y] == map[x-1 + Pw*(y )]) map[x+Pw*y]=0;
        if (map[x+Pw*y] == map[x+1 + Pw*(y )]) map[x+Pw*y]=0;
        if (map[x+Pw*y] == map[x + Pw*(y+1)]) map[x+Pw*y]=0;
        if (map[x+Pw*y] == map[x + Pw*(y-1)]) map[x+Pw*y]=0;
    }
}

```

```
for(y=1, limit=0; y < Ph; y++)
```

```
for(x=0; x<Pw; x++)
```

```
{
```



```

        ff = map[x+Pw*y];
        if(ff != 0)
            for(j=0; j<f; j++) for(k=0; k<f; k++)
            {
                if(*(ff+k*Dw+j) == 1) key[limit++] = (ff+k*Dw+j);
            }
    }
}

// open new window and make intial computation for the zoom window's frame
void
Zoomwin(void)
{
    int i, j, k, x, y, cx, cy;
    char *ff;
    float R;
    int width = Pwt*fv+30, height = Pht*fv+30;

    SRGP_end();
    SRGP_begin("Zoom", width, height, 0, FALSE);
    SetInputMode();
    Start_Colors();
    Pw = Pwt; Ph = Pht; f=fv; Dw=Dwt; Dh=Dht;

    xi = XGetImage(d_dpy, d_act.drawable.xid, 4, 4, Dw, Dh,
                  AllPlanes, ZPixmap);
    for(y=0; y<Ph; y++)
    {
        for(x=0; x<Pw; x++)
        {
            cx = x;
            cy = y;
            if((f*(cx+cy*Dw)) < (Dw*Dh))
                map[x+Pw*y] = &xi->data[f*(cx+cy*Dw)];
            g0[y] = 1;
        }
    }
}

// set format for the rectangular projection (Orthogonal and Mercator)
void
Set_Format()
{

```

```

    SRGP_setColor(0);
    SRGP_fillRectangleCoord(28, 75, 753, 444);
    SRGP_setLineWidth(2);
    SRGP_setColor(1);
    SRGP_rectangleCoord(31, 78, 751, 440);
}

// Clear part of window for the new format
void
Set_Format_New()
{
    SRGP_setColor(0);
    SRGP_fillRectangleCoord(28, 75, 753, 444);
}

// get month, day and year from the date stored in the database
void
Get_date(int index)
{
    int year, month, day, ind1, ind2;
    char date[20];

    year = index / 10000;
    ind1 = (index -(year * 10000));
    month= ind1 / 100;
    ind2 = month * 100;
    day = ind1 - ind2;
    year = year + 1900;

    SRGP_setColor(48);
    SRGP_fillRectangleCoord(293, 23, 484, 37);
    SRGP_setColor(49);
    sprintf(date, "%s%2d/%2d/%4d", "Date: ", month, day, year);
    SRGP_text(SRGP_defPoint(320, 25), date);
}

// get date and display it on the zoom window
void
Get_zoom_date(int index)
{
    int year, month, day, ind1, ind2;
    char date[20];
    int Dw1 = (Dw/2) - 100;

```

```

int Dwr = (Dw/2) + 100;

year = index / 10000;
ind1 = (index -(year * 10000));
month= ind1 / 100;
ind2 = month * 100;
day = ind1 - ind2;
year = year + 1900;

SRGP_setColor(48);
SRGP_fillRectangleCoord(Dwl, 10, Dwr, 24);
SRGP_setColor(49);
sprintf(date, "%s%2d/%2d/%4d", "Date: ", month, day, year);
SRGP_text(SRGP_defPoint(Dwl+25, 12), date);
}

// place a marker next to the selected option
void
Put_marker(int yy)
{
    SRGP_setColor(1);
    SRGP_setMarkerSize(4);
    SRGP_setFillStyle(SOLID);
    SRGP_setMarkerStyle(MARKER_SQUARE);
    SRGP_markerCoord(menu+4, yy+5);
}

// clear some menu areas
void
Clear()
{
    int yp = 465;
    SRGP_setColor(0);
    if(menu >= 640)
        SRGP_fillRectangleCoord(menu-2, yp, menu+82, yp+142);
    else
        SRGP_fillRectangleCoord(menu-2, yp, menu+123, yp+142);
    if(inst_area == 1)
    {
        SRGP_fillRectangleCoord(282, yp, 523, yp+140);
        SRGP_fillRectangleCoord(281, 444, 402, 604);
    }
}

```

```
// display options for the Date menu
```

```
void
```

```
Date_opt()
```

```
{
    int xl = 400, yl = 485;
    SRGP_setColor(1);
    SRGP_setLineWidth(2);
    SRGP_lineCoord(xl+120, yl, xl+120, yl+121);
    SRGP_lineCoord(xl, yl, xl+120, yl);
    SRGP_lineCoord(xl, yl, xl, yl+121);
    SRGP_setColor(34);
    SRGP_text(SRGP_defPoint(xl+10, yl+=24), "SingleFrame >");
    SRGP_text(SRGP_defPoint(xl+10, yl+=20), "SingleFrame <");
    SRGP_text(SRGP_defPoint(xl+10, yl+=20), "Reverse");
    SRGP_text(SRGP_defPoint(xl+10, yl+=20), "Stop");
    SRGP_text(SRGP_defPoint(xl+10, yl+=20), "Increase");
    yl=485;
    if(Dt == 1)
        Put_marker(yl+24);
    else if(Dt == 2)
        Put_marker(yl+44);
    else if(Dt == 3)
        Put_marker(yl+64);
    else if(Dt == 4)
        Put_marker(yl+84);
    else if(Dt == 5)
        Put_marker(yl+104);
}
```

```
// retrieve all stored data sets' name from the database and display them as a list on the
```

```
//Data menu option
```

```
void
```

```
Data_opt()
```

```
{
    int i=0, xl=160, yl = 525;
    Var dataname, adata;

    SRGP_setColor(1);
    SRGP_setLineWidth(2);
    SRGP_lineCoord(xl+120, yl, xl+120, yl+81);
    SRGP_lineCoord(xl, yl, xl+120, yl);
    SRGP_lineCoord(xl, yl, xl, yl+81);
}
```

```

SRGP_setColor(34); yl = 529;
Category * datapt = DB->FindCategory("DATATYPE");
SetQuery data = datapt->GetObjects();
while(data.GetVarInc(adata))
{
    dataname = adata["DATATYPE::dname"];
    sprintf(dnames[i++], "%s", pChar(dataname));
    SRGP_text(SRGP_defPoint(xl+10, yl), dnames[i-1]);
    yl += 20;
}
yl = 525;
if(D == 1)
    Put_marker(yl+4);
else if(D == 2)
    Put_marker(yl+24);
else if(D == 3)
    Put_marker(yl+44);
else if(D == 4)
    Put_marker(yl+64);
}

// display all possible projections on the Projection menu, once this menu is activated
void
Proj_opt()
{
    int xl = 40, yl = 485;
    SRGP_setColor(1);
    SRGP_setLineWidth(2);
    SRGP_lineCoord(xl+120, yl, xl+120, yl+121);
    SRGP_lineCoord(xl, yl, xl+120, yl);
    SRGP_lineCoord(xl, yl, xl, yl+121);
    SRGP_setColor(34);
    SRGP_text(SRGP_defPoint(xl+10, yl+=4), "Mercator");
    SRGP_text(SRGP_defPoint(xl+10, yl+=20), "Homolographic");
    SRGP_text(SRGP_defPoint(xl+10, yl+=20), "Stereographic");
    SRGP_text(SRGP_defPoint(xl+10, yl+=20), "Sinusoidal");
    SRGP_text(SRGP_defPoint(xl+10, yl+=20), "Orthogonal");
    SRGP_text(SRGP_defPoint(xl+10, yl+=20), "Orthographic");
    yl=485;
    if(P == 1)
        Put_marker(yl+4);
    else if(P == 2)
        Put_marker(yl+24);
}

```

```

    else if(P == 3)
        Put_marker(yl+44);
    else if(P == 4)
        Put_marker(yl+64);
    else if(P == 5)
        Put_marker(yl+84);
    else if(P == 6)
        Put_marker(yl+104);
}

// display possible options for the Rotation menu
void
Rot_opt()
{
    int xl = 280, yl = 545;
    SRGP_setColor(1);
    SRGP_setLineWidth(2);
    SRGP_lineCoord(xl+120, yl, xl+120, yl+61);
    SRGP_lineCoord(xl, yl, xl+120, yl);
    SRGP_lineCoord(xl, yl, xl, yl+61);
    SRGP_setColor(34);
    SRGP_text(SRGP_defPoint(xl+10, yl+=4), "Reduce");
    SRGP_text(SRGP_defPoint(xl+10, yl+=20), "Stop");
    SRGP_text(SRGP_defPoint(xl+10, yl+=20), "Increase");
    yl=545;
    if(R == 1)
        Put_marker(yl+4);
    else if(R == 2)
        Put_marker(yl+24);
    else if(R == 3)
        Put_marker(yl+44);
}

// display options for the Cloud menu
void
Cloud_opt()
{
    int xl = 520, yl = 545;
    SRGP_setColor(1);
    SRGP_setLineWidth(2);
    SRGP_lineCoord(xl+120, yl, xl+120, yl+61);
    SRGP_lineCoord(xl, yl, xl+120, yl);
    SRGP_lineCoord(xl, yl, xl, yl+61);

```

```

    SRGP_setColor(34);
    SRGP_text(SRGP_defPoint(xl+10, yl+=22), "Visible");
    SRGP_text(SRGP_defPoint(xl+10, yl+=20), "NonVisible");
    yl=545;
    if(Cl == 1)
        Put_marker(yl+24);
    else if(Cl == 2)
        Put_marker(yl+44);
}

// display options for the zoom menu
void
Zoom_opt()
{
    int xl = 640, yl = 465;

    SRGP_setColor(1);
    SRGP_setLineWidth(2);
    SRGP_lineCoord(xl+80, yl, xl+80, yl+141);
    SRGP_lineCoord(xl, yl, xl+80, yl);
    SRGP_lineCoord(xl, yl, xl, yl+141);
    SRGP_setColor(34);
    SRGP_text(SRGP_defPoint(xl+10, yl+=4), "120%");
    SRGP_text(SRGP_defPoint(xl+10, yl+=20), "100%");
    SRGP_text(SRGP_defPoint(xl+10, yl+=20), "80%");
    SRGP_text(SRGP_defPoint(xl+10, yl+=20), "60%");
    SRGP_text(SRGP_defPoint(xl+10, yl+=20), "40%");
    SRGP_text(SRGP_defPoint(xl+10, yl+=20), "20%");
    SRGP_text(SRGP_defPoint(xl+10, yl+=20), "End");
    yl=465;
    if(Zp == 1)
        Put_marker(yl+4);
    else if(Zp == 2)
        Put_marker(yl+24);
    else if(Zp == 3)
        Put_marker(yl+44);
    else if(Zp == 4)
        Put_marker(yl+64);
    else if(Zp == 5)
        Put_marker(yl+84);
    else if(Zp == 6)
        Put_marker(yl+104);
}

```



```

// set marker on the non-visible menu options
void
set_marker()
{
    SRGP_setColor(1);
    SRGP_setMarkerSize(4);
    SRGP_setMarkerStyle(MARKER_SQUARE);
}

// make a query to the database to retrieve all possible satellites for the selected data set
//and display them on the menu list
void
Satel_opt(char *dname)
{
    int i=0, xl = 282, yl = 545;
    Var sateln;

    SRGP_setColor(1);
    SRGP_setLineWidth(2);
    SRGP_lineCoord(xl+120, yl, xl+120, yl+42);
    SRGP_lineCoord(xl, yl, xl+120, yl);
    SRGP_lineCoord(xl, yl+42, xl+120, yl+42);
    SRGP_setColor(34); yl = 549;
    Var S = RangeQuery(TheRelation("DATATYPE::dname", DB), dname);
    SetQuery Satel = S(TheRelation("DATATYPE::takeby", DB));
    while(Satel.GetVarInc(sateln))
    {
        Var SName = sateln["Satellite::sname"];
        sprintf(satelnname[i++], "%s", pChar(SName));
        SRGP_text(SRGP_defPoint(xl+10, yl), satelnname[i-1]);
        yl += 20;
    }
    yl = 545;
    if(satellite == 1)
    {
        set_marker();
        SRGP_markerCoord(282+4, yl+9);
    }
    else if(satellite == 2)
    {
        set_marker();
        SRGP_markerCoord(282+4, yl+29);
    }
}

```

```

}

// retrieve all possible instruments from the database and display them on the menu option
//box
void
inst_opt(char *sateln)
{
    int i=0, xl=282, yl=445;
    Var insts;

    SRGP_setColor(0);
    SRGP_fillRectangleCoord(281, 444, 402, 604);
    SRGP_setColor(1);
    SRGP_setLineWidth(2);
    SRGP_lineCoord(xl+120, yl, xl+120, 604);
    SRGP_lineCoord(xl, yl, xl+120, yl);
    SRGP_lineCoord(xl, 604, xl+120, 604);
    SRGP_lineCoord(xl, yl, xl, yl+100);
    SRGP_setColor(34); yl = 447;
    Var S = RangeQuery(TheRelation("Satellite::sname", DB), sateln);
    SetQuery Obser = S(TheRelation("Satellite::uses", DB));
    while(Obser.GetVarInc(insts))
    {
        Var instused = insts["Instrument::iname"];
        sprintf(instrs[i++], "%s", pChar(instused));
        SRGP_text(SRGP_defPoint(xl+10, yl), instrs[i-1]);
        yl += 20;
    }
    yl = 445;
    if(inst == 1)
    {
        set_marker();
        SRGP_markerCoord(xl+4, yl+7);
    }
    else if(inst == 2)
    {
        set_marker();
        SRGP_markerCoord(xl+4, yl+27);
    }
    else if(inst == 3)
    {
        set_marker();
        SRGP_markerCoord(xl+4, yl+47);
    }
}

```

```

}
else if(inst == 4)
{
    set_marker();
    SRGP_markerCoord(xl+4, yl+67);
}
else if(inst == 5)
{
    set_marker();
    SRGP_markerCoord(xl+4, yl+87);
}
else if(inst == 6)
{
    set_marker();
    SRGP_markerCoord(xl+4, yl+107);
}
else if(inst == 7)
{
    set_marker();
    SRGP_markerCoord(xl+4, yl+127);
}
else if(inst == 8)
{
    set_marker();
    SRGP_markerCoord(xl+4, yl+147);
}
}

```

// retrieve possible frequencies from the database and display them on the screen
void

DataFreq_opt(char *instn)

```

{
    int i=0, xl = 402, yl = 545;
    Var pfreq;

    SRGP_setColor(1);
    SRGP_setLineWidth(2);
    SRGP_lineCoord(xl+120, yl, xl+120, yl+42);
    SRGP_lineCoord(xl, yl, xl+120, yl);
    SRGP_lineCoord(xl, yl+42, xl+120, yl+42);
    SRGP_setColor(34); yl = 549;
    Var S = RangeQuery(TheRelation("Instrument::iname", DB), instn);
    SetQuery Obser = S(TheRelation("Instrument::progused", DB));
}

```

```

while(Obser.GetVarInc(pfreq))
{
    Var obserfreq = pfreq["ObserProg::frequency"];
    sprintf(freqs[i++], "%s", pChar(obserfreq));
    SRGP_text(SRGP_defPoint(xl+10, yl), freqs[i-1]);
    yl += 20;
}
yl = 545;
if(freq == 1)
{
    set_marker();
    SRGP_markerCoord(402+4, yl+9);
}
else if(freq == 2)
{
    set_marker();
    SRGP_markerCoord(402+4, yl+29);
}
}

// determine what action to take depending on the coordinates of the mouse click
int
Find_action(point pos)
{
    if(pos.y >= 610 && pos.y <= 650)
    {
        if(menu != 0) Clear();
        satel_area = 0;
        freq_area = 0;
        if(pos.x >= 40 && pos.x <= 120) //projections
        {
            menu = 40;
            Proj_opt();
            area = 1;
            return 0;
        }
        else if(pos.x >= 160 && pos.x <= 240) //data
        {
            menu=160;
            Data_opt();
            area=2;
            return 0;
        }
    }
}

```

```

else if(pos.x >= 280 && pos.x <= 360) //rotation
{
    menu = 280;
    Rot_opt();
    area=3;
    return 0;
}
else if(pos.x >= 400 && pos.x <= 480) //Date
{
    menu = 400;
    Date_opt();
    area=4;
    return 0;
}
else if(pos.x >= 760 && pos.x <= 840) // exit
    return 7;
else if(pos.x >= 520 && pos.x <= 600) // cloud
{
    menu = 520;
    Cloud_opt();
    area=5;
    return 0;
}
else if(pos.x >= 640 && pos.x <= 720) // zoom
{
    menu = 640;
    Zoom_opt();
    area=6;
    return 0;
}
}
if(pos.x >= 40 && (pos.x <= 160) && (area == 1)) // projections opt
{
    if(pos.y >= 485 && pos.y <= 508)
    {
        Clear(); P=1;
        Proj_opt();
        return 9;
    }
    else if(pos.y >= 509 && pos.y <= 528)
    {
        Clear(); P=2;
        Proj_opt();
    }
}

```

```

        return 11;
    }
    else if(pos.y >= 529 && pos.y <= 548)
    {
        Clear(); P=3;
        Proj_opt();
        return 12;
    }
    else if(pos.y >= 549 && pos.y <= 568)
    {
        Clear(); P=4;
        Proj_opt();
        return 14;
    }
    else if(pos.y >= 569 && pos.y <= 588)
    {
        Clear(); P=5;
        Proj_opt();
        return 15;
    }
    else if(pos.y >= 589 && pos.y <= 608)
    {
        Clear(); P=6;
        Proj_opt();
        return 16;
    }
}
if((pos.x >= 160) && (pos.x <= 279) && (area==2)) //data opt
{
    if(pos.y >= 525 && pos.y <= 544)
    {
        Clear(); D=1;
        Data_opt();
        dindex = 0; satellite=0;
        Satel_opt(dnames[0]); satel_area=1; return 0;
    }
    else if(pos.y >= 545 && pos.y <= 568)
    {
        Clear(); D=2;
        Data_opt();
        dindex = 1; satellite=0;
        Satel_opt(dnames[1]);
        satel_area=1; return 0;
    }
}

```

```

}
else if(pos.y >= 569 && pos.y <= 588)
{
    Clear(); D=3;
    Data_opt();
    dindex = 2; satellite=0;
    Satel_opt(dnames[2]);
    satel_area=1; return 0;
}
else if(pos.y >= 589 && pos.y <= 608)
{
    Clear(); D=4;
    Data_opt();
    dindex = 3; satellite=0;
    Satel_opt(dnames[3]);
    satel_area = 1;
    return 0;
}
}
// satellite option
iff(pos.x >= 280) && (pos.x <= 399) && (satel_area == 1) && (area == 2))
{
    iff(pos.y >= 545 && pos.y <= 568)
    {
        SRGP_setColor(0);
        SRGP_fillRectangleCoord(282, 544, 402, 544+44);
        satellite = 1;
        inst = 0; sindex=0;
        SRGP_fillRectangleCoord(402, 544, 522, 544+44);
        Satel_opt(dnames[dindex]);
        inst_opt(satelname[0]);
        satel_area = 0;
        inst_area = 1;
        return 0;
    }
    else if(pos.y >= 569 && pos.y <= 588)
    {
        SRGP_setColor(0);
        SRGP_fillRectangleCoord(282, 544, 402, 544+44);
        satellite = 2;
        inst = 0; sindex=1;
        SRGP_fillRectangleCoord(402, 544, 522, 544+44);
        Satel_opt(dnames[dindex]);
    }
}

```

```

    inst_opt(satelname[1]);
    satel_area = 0;
    inst_area = 1;
    return 0;
}
}
// instrument option
if((pos.x >= 280) && (pos.x <= 399) && (inst_area == 1) && (area == 2))
{
    if(pos.y >= 445 && pos.y <= 468)
    {
        SRGP_setColor(0);
        SRGP_fillRectangleCoord(282, 444, 402, 604);
        SRGP_fillRectangleCoord(402, 544, 522, 544+44);
        inst = 1; freq = 0; iindex=0;
        inst_opt(satelname[sindex]);
        DataFreq_opt(instrs[0]);
        freq_area=1;
        return 0;
    }
    else if(pos.y >= 469 && pos.y <= 488)
    {
        SRGP_setColor(0);
        SRGP_fillRectangleCoord(282, 444, 402, 604);
        SRGP_fillRectangleCoord(402, 544, 522, 544+44);
        inst = 2; freq = 0; iindex=1;
        inst_opt(satelname[sindex]);
        DataFreq_opt(instrs[1]);
        freq_area = 1;
        return 0;
    }
    else if(pos.y >= 489 && pos.y <= 508)
    {
        SRGP_setColor(0);
        SRGP_fillRectangleCoord(282, 444, 402, 604);
        SRGP_fillRectangleCoord(402, 544, 522, 544+44);
        inst = 3; freq = 0; iindex=2;
        inst_opt(satelname[sindex]);
        DataFreq_opt(instrs[2]);
        freq_area = 1;
        return 0;
    }
    else if(pos.y >= 509 && pos.y <= 528)

```



```

{
    SRGP_setColor(0);
    SRGP_fillRectangleCoord(282, 444, 402, 604);
    SRGP_fillRectangleCoord(402, 544, 522, 544+44);
    inst = 4; freq = 0; iindex = 3;
    inst_opt(satelname[sindex]);
    DataFreq_opt(instrs[3]); freq_area = 1;
    return 0;
}
else if(pos.y >= 529 && pos.y <= 548)
{
    SRGP_setColor(0);
    SRGP_fillRectangleCoord(282, 444, 402, 604);
    SRGP_fillRectangleCoord(402, 544, 522, 544+44);
    inst = 5; freq = 0; iindex = 4;
    inst_opt(satelname[sindex]);
    DataFreq_opt(instrs[4]);
    freq_area = 1;
    return 0;
}
else if(pos.y >= 549 && pos.y <= 568)
{
    SRGP_setColor(0);
    SRGP_fillRectangleCoord(282, 444, 402, 604);
    SRGP_fillRectangleCoord(402, 544, 522, 544+44);
    inst = 6; freq = 0; iindex = 5;
    inst_opt(satelname[sindex]);
    DataFreq_opt(instrs[5]);
    freq_area = 1;
    return 0;
}
else if(pos.y >= 569 && pos.y <= 588)
{
    SRGP_setColor(0);
    SRGP_fillRectangleCoord(282, 444, 402, 604);
    SRGP_fillRectangleCoord(402, 544, 522, 544+44);
    inst = 7; freq = 0; iindex = 6;
    inst_opt(satelname[sindex]);
    DataFreq_opt(instrs[6]); freq_area = 1;
    return 0;
}
else if(pos.y >= 589 && pos.y <= 608)
{

```

```

        SRGP_setColor(0);
        SRGP_fillRectangleCoord(282, 444, 402, 604);
        SRGP_fillRectangleCoord(402, 544, 522, 544+44);
        inst = 8; freq = 0; iindex = 7;
        inst_opt(satelname[sindex]);
        DataFreq_opt(instrs[7]);
        freq_area = 1;
        return 0;
    }
}
// frequency options
if((pos.x >= 400) && (pos.x <= 519) && (freq_area == 1) && (area== 2))
{
    if(pos.y >= 545 && pos.y <= 568)
    {
        SRGP_setColor(0);
        SRGP_fillRectangleCoord(404, 544, 522, 544+44);
        freq = 1; findex = 0;
        DataFreq_opt(instrs[iindex]); Dc = 1;
        return 18;
    }
    else if(pos.y >= 569 && pos.y <= 588)
    {
        SRGP_setColor(0);
        SRGP_fillRectangleCoord(404, 544, 522, 544+44);
        freq = 2; findex = 1;
        DataFreq_opt(instrs[iindex]); Dc = 1;
        return 18;
    }
}
if((pos.x >= 280) && (pos.x <= 399) && (area==3)) // rotation opt
{
    if(pos.y >= 545 && pos.y <= 568)
    {
        Clear(); R=1;
        Rot_opt();
        return 1;
    }
    else if(pos.y >= 569 && pos.y <= 588)
    {
        Clear(); R=2;
        Rot_opt();
        return 2;
    }
}

```

```

    }
    else if(pos.y >= 589 && pos.y <= 608);
    {
        Clear(); R=3;
        Rot_opt();
        return 3;
    }
}
if((pos.x >=400) && (pos.x <=519) && (area==4)) // date opt
{
    if(pos.y >= 509 && pos.y <= 528)
    {
        Clear(); Dt=1;
        Date_opt();
        return 22;
    }
    else if(pos.y >= 529 && pos.y <= 548)
    {
        Clear(); Dt=2;
        Date_opt();
        return 23;
    }
    else if(pos.y >= 549 && pos.y <= 568)
    {
        Clear(); Dt=3;
        Date_opt();
        return 6;
    }
    else if(pos.y >= 569 && pos.y <= 588)
    {
        Clear(); Dt=4;
        Date_opt();
        return 4;
    }
    else if(pos.y >= 589 && pos.y <= 608)
    {
        Clear(); Dt=5;
        Date_opt();
        return 5;
    }
}
if((pos.x >= 520) && (pos.x <= 600) && (area==5)) // cloud opt
{

```

```

    if(pos.y>= 569 && pos.y <= 588)
    {
        Clear(); Cl=1;
        Cloud_opt();
        return 19;
    }
    else if(pos.y >= 589 && pos.y <= 608)
    {
        Clear(); Cl=2;
        Cloud_opt();
        return 20;
    }
}
if((pos.x >= 640) && (pos.x <= 720) && (area==6)) // Zoom opt
{
    if(pos.y >= 465 && pos.y <= 488)
        return 31;
    else if(pos.y >= 489 && pos.y <= 508)
        return 30;
    else if(pos.y >= 509 && pos.y <= 528)
        return 29;
    else if(pos.y >= 529 && pos.y <= 548)
        return 28;
    else if(pos.y >= 549 && pos.y <= 568)
        return 27;
    else if(pos.y >= 569 && pos.y <= 588)
        return 26;
    else if(pos.y >= 589 && pos.y <= 608)
    {
        Clear();
        Zo=1; Zp=6;
        Zoom_opt();
        return 25;
    }
}
if(pos.y >= 20 && pos.y <= 42)
{
    if(pos.x >= 290 && pos.x <= 491)
    {
        See_date = 0; return 0;
    }
    else if(pos.x >= 493 && pos.x <= 512)
    {

```

```

        See_date = 1; return 24;
    }
    else if(pos.x >= 267 && pos.x <= 286)
    {
        See_date = 2; return 17;
    }
}
Clear();
return 0;
}

// get mouse click point and take the right action
void
Get_action(int bt, const long int rdate[], const char exts[])
{
    int dev, action;
    char com[4];
    locator measure[100];

    dev = SRGP_waitEvent(1);
    if(dev == LOCATOR)
    {
        SRGP_getLocator(measure);
        if((measure->button_chord[LEFT_BUTTON]) == UP)
        {
            action = Find_action(measure->position);
            switch(action)
            {
                case 1 : if(Rot<0) Rot += 1; break;
                case 2 : Rot = 0; break;
                case 3 : Rot -= 1; break;
                case 4 : vd = 0; break;
                case 5 : vd = 1; Sf=0; break;
                case 6 : vd = 2; Sf=0; break;
                case 7 : exit(1); break;
                case 9 : if(opt!=0)
                    {
                        opt = 0;
                        Set_Format();
                        Startup(bt, exts);
                    } break;
                case 11 : if(opt!=3)

```

```

        {
            opt = 3;
            Set_Format_New();
            Startup(bt, exts);
        } break;
case 12 : if(opt!=4)
        {
            opt = 4;
            Set_Format_O();
            Startup(bt, exts);
        } break;
case 14 : if(opt!=6)
        {
            opt = 6;
            Set_Format_New();
            Startup(bt, exts);
        } break;
case 15 : if(opt!=7)
        {
            opt = 7;
            Set_Format();
            Startup(bt, exts);
        } break;
case 16 : if(opt!=8)
        {
            opt = 8;
            Set_Format_O();
            Startup(bt, exts);
        } break;
case 17 : if(indrd <= 0) indrd = max_ind;
            indrd -= 1;
            Get_date(rdate[indrd]);
            break;
case 18 : TorO = 1;
            if(opt == 0 || opt == 7)
                Set_Format();
            else if(opt == 4 || opt == 8)
                Set_Format_O();
            else if(opt == 3 || opt == 6)
                Set_Format_New();
            Startup(bt, exts);
            break;
case 19 : flag = 0; break;

```

```

case 20 : flag = 1; break;
case 22 : Sf = 1; vd = 1; break;
case 23 : Sf = 1; vd = 2; break;
case 24 : if(indrd > max_ind) indrd = -1;
          indrd += 1;
          Get_date(rdate[indrd]);
          break;
case 25 : Zo = 1; Zoom = 0;
          Pwt=360; Pht=180; Dwt=720; Dht=360; fv=2;
          Set_Format_New();
          Startup(bt, exts);
          break;
case 26 : prev_fv = fv; fv = 2;
          Clear(); Zp=6; Zoom_opt();
          if(Zoom == 1)
          {
              if((Pwt*f_v) < 721 && (Pht*f_v) < 361)
              {
                  Clear(); Zp=6;
                  Zoom_opt();
                  Dwt = Pwt*f_v; Dht = Pht*f_v;
                  Set_Format_New();
                  Startup(bt, exts);
              }
              else
                  fv = prev_fv;
          } break;
case 27 : prev_fv = fv; fv = 4;
          Clear(); Zp=5;
          Zoom_opt();
          if(Zoom == 1)
          {
              if((Pwt*f_v) < 721 && (Pht*f_v) < 361)
              {
                  Clear(); Zp=5;
                  Zoom_opt();
                  Dwt = Pwt*f_v; Dht = Pht*f_v;
                  Set_Format_New();
                  Startup(bt, exts);
              }
              else
                  fv = prev_fv;
          } break;

```

```

case 28 : prev_fv = fv; fv = 6;
        Clear(); Zp=4;
        Zoom_opt();
        if(Zoom == 1)
        {
            if((Pwt*fv) < 721 && (Pht*fv) < 361)
            {
                Clear(); Zp=4; Zoom_opt();
                Dwt = Pwt*fv; Dht = Pht*fv;
                Set_Format_New();
                Startup(bt, exts);
            }
            else
                fv = prev_fv;
        } break;
case 29 : prev_fv = fv; fv = 8;
        Clear(); Zp=3;
        Zoom_opt();
        if(Zoom == 1)
        {
            if((Pwt*fv) < 721 && (Pht*fv) < 361)
            {
                Clear(); Zp=3;
                Zoom_opt();
                Dwt = Pwt*fv; Dht = Pht*fv;
                Set_Format_New();
                Startup(bt, exts);
            }
            else
                fv = prev_fv;
        } break;
case 30 : prev_fv = fv; fv = 10;
        Clear(); Zp = 2;
        Zoom_opt();
        if(Zoom == 1)
        {
            if((Pwt*fv) < 721 && (Pht*fv) < 361)
            {
                Clear(); Zp = 2;
                Zoom_opt();
                Dwt = Pwt*fv; Dht = Pht*fv;
                Set_Format_New();
                Startup(bt, exts);
            }
        }

```



```

        }
        else
            fv = prev_fv;
    } break;
case 31 : prev_fv = fv;  fv = 12;
Clear(); Zp = 1;
Zoom_opt();
if(Zoom == 1)
{
    if((Pwt*fv) < 721 && (Pht*fv) < 361)
    {
        Clear(); Zp = 1;
        Zoom_opt();
        Dwt = Pwt*fv; Dht = Pht*fv;
        Set_Format_New();
        Startup(bt, exts);
    }
    else
        fv = prev_fv;
} break;
default: break;
}
}
else if((measure->button_chord[LEFT_BUTTON] == DOWN) && ( area==6))
{
    if((measure->position.y >= 80) && (measure->position.y <= 440) &&
        (measure->position.x < 752))
    {
        initxl = ((measure->position.x) - 30)/2;
        inityl = ((measure->position.y) - 80)/2;
        SRGP_setColor(1);
        SRGP_setMarkerSize(6);
        SRGP_setMarkerStyle(MARKER_CIRCLE);
        SRGP_markerCoord(measure->position.x, measure->position.y);
        SRGP_waitEvent(-1);
        SRGP_getLocator(measure);
        initxr = ((measure->position.x) - 30)/2;
        inityr = ((measure->position.y) - 80)/2;
        if((initxr-initxl) > 3 && (inityl-inityr) > 3 )
        {
            if(fv == 12) Zp = 1;
            else if(fv == 10) Zp = 2;
            else if(fv == 8) Zp = 3;

```

```

else if(fv == 6) Zp = 4;
else if(fv == 4) Zp = 5;
else if(fv == 2) Zp = 6;
Clear(); Zoom_opt();
Set_Format_New();
Pwt = initxr - initxl;
Pht = inityl - inityr;
if(Pwt & 1)
    Pwt += 1;
if(initxr & 1) initxr += 1;
if(initxl & 1) initxl += 1;
if(Pht & 1)
    Pht += 1;
if(inityl & 1) inityl += 1;
if(inityr & 1) inityr += 1;
Dwt = Pwt * fv;
Dht = Pht * fv;
if(bt == 360)
{
    inityl= 180 - inityl;
    inityr= 180 - inityr;
}
Zoom = 1;
Zoomwin(); cloud = 0;
}
else
{
    Clear(); fv = 2;
    Zo = 1; Zp = 4;
    Zoom_opt();
}
Zcheck = 0;
}
}
}
else if(dev == KEYBOARD) // keyboard actions
{
    SRGP_getKeyboard(com, 4);
    switch(com[0])
    {
        /* ... menu options ... */
        case 'p' : menu=40; Proj_opt(); break;
        case 'd' : menu=160; Data_opt(); break;
    }
}

```

```

        case 'r' : menu=280; Rot_opt(); break;
        case 't' : menu=400; Date_opt(); break;
        case 'c' : menu=520; Cloud_opt(); break;
        case 'z' : menu=640; Zoom_opt(); break;
        case 'e' : exit(1); break;
        default : break;
    }
}
}

// find action from the zoom window
int
Find_zoom_action(point pos)
{
    return 1;
}

// tack mouse movement in the zoom window and return to the main program window
void
Get_zoom_action(int bt, const long int rdate[], const char exts[])
{
    int dev, action;
    char com[4];
    locator measure[100];

    dev = SRGP_waitEvent(1);
    if(dev == LOCATOR)
    {
        SRGP_getLocator(measure);
        action = Find_zoom_action(measure->position);
        switch(action)
        {
            case 1 : Dc=1; Zo = 1; opt = 7; Zoom = 0;
                    SRGP_end();
                    SRGP_begin("FIU High Performance Database Research Center",
                                860,660,0,FALSE);
                    Pwt=360; Pht=180; Dwt=720; Dht=360; fv=2; Zp=6;
                    Start_Colors();
                    Set_Menu();
                    SetInputMode();
                    Set_Format_New();
                    Startup(bt, exts);

```

```

        break;
    default: break;
}
}
}

```

// display frame in the zoom window for data sets that have 576 bytes per line

void

Zoom_Ozone(int index, Var in_pt, int bts, const char ext[])

```

{
    int x, yy, y, tp, j, i, k, n, zx, zzy, zy=0;
    unsigned char b[576];
    char *ff, *gg;
    int r[360], or[360], tmpm[360];

    for (x=0; x<Pw; x++)
    {
        r[x] = (x+fn+Pw)%Pw;
        or[x] = (x+Rot+Pw)%Pw;
    }
    fn = (fn+Rot+Pw) % Pw;

    for (y=0; y<180; y++)
    {
        n = fastread(b, bts, 1);

        if((y <= inityl) && (y > inityr))
        {
            zx = 0;
            for (x=0; x<360; x++)
            {
                if((x >= initxl) && (x < initxr))
                {
                    n = 2*((4*x)/5);
                    yy = 179 - y;
                    if (cmap[x+360*yy] == '0')
                        tp=color[256*b[n]+b[n+1]];
                    else tp = 177;
                    zzy = Ph - zy;
                    if(cloud*tp == 42)
                        tp = pcol[or[zx]+Pw*zzy];
                    tmpm[zx] = tp;
                    ff = map[zx+Pw*zzy];
                }
            }
        }
    }
}

```

```

        zx += 1;
        if(ff != 0)
            if(tp != *ff)
                for(j=0; j<f; j++)
                    for(k=0; k<f; k++) *(ff+k*Dw+j)=tp;
    }
}
for(x=0; x<Pw; x++)
    pcol[x+Pw*zy] = tmpm[x];
zy += 1;
}
}
cloud = flag;
send_zoom_frame();
if(vd != 0) Get_zoom_date(index);
}

```

// display frame in the zoom window for data sets that have 360 bytes per line

void

Zoom_frame(int index, Var in_pt, int bts, const char ext[])

```

{
    int x, yy, y, tp, j, i, k, n, zx, zy=0;
    unsigned char b[576];
    char *ff, *gg;
    int r[360], or[360], tmpm[360];

    for (x=0; x<Pw; x++)
    {
        r[x] = (x+fn+Pw)%Pw;
        or[x] = (x+Rot+Pw) % Pw;
    }
    fn = (fn+Rot+Pw) % Pw;
    for (y=0; y<180; y++)
    {
        n = fastread(b,bts,1);

        if((y >= inityl) && (y < inityr))
        {
            zx = 0;
            for (x=0; x<360; x++)
            {
                if((x >= initxl) && (x < initxr))
                {

```

```

        tp=color[b[x]];
        if(cloud*tp == 42)
            tp = pcol[or[zx]+Pw*zy];
        if((bts == 360) && (ext[0] == 'c') && (cloud == 0))
            if((tp == 42) && (cmap[x+360*y] == '0'))
                tp = 50;
            tmpm[zx] = tp;

        ff = map[zx+Pw*zy];
        zx += 1;
        if(ff != 0)
            if(tp != *ff)
                for(j=0; j<f; j++)
                    for(k=0; k<f; k++) *(ff+k*Dw+j)=tp;
    }
}
for(x=0; x<Pw; x++)
    pcol[x+Pw*zy] = tmpm[x];
zy += 1;
}
}
cloud = flag;
send_zoom_frame();
if(vd != 0) Get_zoom_date(index);
}

```

// get data from the buffer, one line at a time, process it, compose the whole frame and
//send it to the screen

void

Next_Frame(int index, Var in_pr, int bts, const char exts[])

```

{
    int x, yy, y, tp, j, i, k, n;
    unsigned char b[576];
    char *ff, *gg;
    int r[360], or[360], tmpm[360];

    for (x=0; x<360; x++)
    {
        r[x] = (x+fn+360)%360;
        or[x] = (x+Rot+360) % 360;
    }
    fn = (fn+Rot+360) % 360;

```

```

for (y=0; y<180; y++)
{
    n = fastread(b,bts,1);

    for (x=0; x<360; x++)
    {
        if(bts == 360)
        {
            n = r[x]; yy = y;
            tp=color[b[n]];
        }
        else
        {
            n = 2*((4*r[x])/5); yy = 179 - y;
            if (cmap[r[x]+360*yy] == '0')
                tp=color[256*b[n]+b[n+1]];
            else tp = 177;
        }
        if(cloud*tp == 42)
            tp = pcol[or[x]+360*yy];
        if((bts == 360) && (exts[0] == 'c') && (cloud == 0))
            if((tp == 42) && (cmap[r[x]+360*y] == '0'))
                tp = 50;
        tmpm[x] = tp;
        ff = map[x+Pw*yy];

        switch(opt)
        {
            case 0: //Mercantor
                if( ff != 0 ) if (tp != *ff)
                    if (bts == 576 || exts[0] == 'c')
                        for(j=0; j<f; j++)
                            for(k=0; k<g1[yy]; k++) *(ff+k*Dw+j)=tp;
                    else
                        for(j=0; j<k; j++)
                            for(k=0; k<f*g0[yy]; k++) *(ff+k*Dw+j)=tp;
                break;

            default:
                if(ff != 0)
                    if(tp != *ff)
                        for(j=0; j<f; j++)
                            for(k=0; k<f; k++) *(ff+k*Dw+j)=tp;
        }
    }
}

```

```

        break;
    }
}
for(x=0; x<360; x++)
    pcol[x+360*yy] = tmpm[x];
}

if(bts == 360)
    for(i=0; i<limit; i++) *(key[i]) = color[41];
else
    for(i=0; i<limit; i++) *(key[i]) = color[650];

cloud = flag;
send_frame();
if(vd != 0) Get_date(index);
}

// perform all major queries to the database and control users interactions
void
Show(TDataBase* DB)
{
    long int Rdates[2000];
    char datekey[100], pdescr[100];
    Sort<long int> datesort;
    Var fq, sdate, edate, rd, obserp, prgdesc, insts, instru, Rdate;
    int ind, i, x, range, check=0, check1=0, check2=0;
    Var in_pt, Satels, Info, ext, bt, table, mapcols, Image, instname;

    iotp = time(tp); itp = time(tp); printf("Time %d\n",itp-iotp);
    while(1)
    {
        if(Dc == 1)
        {
            SRGP_setColor(0);
            SRGP_fillRectangleCoord(773, 60,857,542);
            SRGP_fillRectangleCoord(30, 445, 260, 465);
            SRGP_fillRectangleCoord(740, 555, 858, 590);
            Info = RangeQuery(TheRelation("DATATYPE::dname", DB), dnames[dindex]);
            ext = Info["DATATYPE::cmext"];
            bt = Info["DATATYPE::bytes"];
            table = Info["DATATYPE::hastable"];
            mapcols = table["ColorTable::mapcolor"];
            Fptr = 0;

```



```

dbread(FileBuff, 1, sizeof(FileBuff), mapcols);
fastread(color, 1304, 1);
dbclose(mapcols);
Color_Chart(dnames[dindex]);
ind = 8700;
sprintf(datekey, "%li%s", ind, pChar(ext));
Image = RangeQuery(TheRelation("BITMAP::date",DB), datekey);
in_pt = Image["BITMAP::image"];
Fptr = 0;
dbread(FileBuff, 1, sizeof(FileBuff), in_pt);
fastread(cmap, 64800, 1);
dbclose(in_pt);
cloud = 0;
instru=RangeQuery(TheRelation("Instrument::iname", DB), instrs[iindex]);
SetQuery Obser = instru(TheRelation("Instrument::progused", DB));
while(Obser.GetVarInc(oberp))
{
    fq = oberp["ObserProg::frequency"];
    if(strcmp(pChar(fq), freqs[findex]) == 0)
    {
        prgdesc = oberp["ObserProg::datadescr"];
        sdate = oberp["ObserProg::startdate"];
        edate = oberp["ObserProg::enddate"];
        break;
    }
}
SRGP_setColor(1);
sprintf(pdescr, "%s", pChar(prgdesc));
SRGP_text(SRGP_defPoint(30, 445), pdescr);
SetQuery bdata = oberp(TheRelation("ObserProg::hasdata", DB));
range = 0;
while(bdata.GetVarInc(Image))
{
    rd = Image["BITMAP::realdate"];
    Rdates[range++] = Long(rd);
}
max_ind = range;
datesort.InsertionSort(Rdates, range);
}
for(indrd = 0; indrd < range; indrd++)
{
    SetQuery dataf = oberp(TheRelation("ObserProg::hasdata", DB));
    while(dataf.GetVarInc(Image) && check == 0)

```

```

{
    Rdate = Image["BITMAP::realdate"];
    if(Long(Rdate) == Rdates[indrd])
    {
        in_pt = Image["BITMAP::image"];
        check = 1;
    }
}
Fptr = 0; Dc = 0; check = 0;
dbread(FileBuff, 1, sizeof(FileBuff), in_pt);
if(Zoom == 0)
{
    Next_Frame(Long(Rdate), in_pt, Long(bt), pChar(ext));
    Get_action(Long(bt), Rdates, pChar(ext));
}
else if((Zoom == 1) && (Long(bt) == 360))
{
    Zoom_frame(Long(Rdate), in_pt, Long(bt), pChar(ext));
    Get_zoom_action(Long(bt), Rdates, pChar(ext));
}
else if((Zoom == 1) && (Long(bt) == 576))
{
    Zoom_Ozone(Long(Rdate), in_pt, Long(bt), pChar(ext));
    Get_zoom_action(Long(bt), Rdates, pChar(ext));
}
dbclose(in_pt);
if(See_date == 1 || See_date == 2)
{
    while(See_date == 1)
        Get_action(Long(bt), Rdates, pChar(ext));
    while(See_date == 2)
        Get_action(Long(bt), Rdates, pChar(ext));
}
if(freq == 0)
    while(freq == 0) Get_action(Long(bt), Rdates, pChar(ext));
if((vd == 0) && (Rot == 0))
    while((vd == 0) && (Rot == 0))
        Get_action(Long(bt), Rdates, pChar(ext));
if(Sf != 0)
{
    vd = 0;
    while(vd == 0)
        Get_action(Long(bt), Rdates, pChar(ext));
}

```

```

    }
    if(vd == 0) ind -= 1;
    else if(vd == 2) indrd -= 2;
    if(indrd < 0) indrd = range-2;
    itp = time(tp); printf("Time %d\n",itp-iotp); iotp = itp;
    if(Dc != 0) break;
}
}
}

int
main(void)
{
    SRGP_begin("FIU High Performance Database Research Center",860,660,0,FALSE);
    Start_Colors();
    Set_Menu();
    memset(map, 0, sizeof(map));
    SetInputMode();
    DB = OpenDataBase("/home/hpdrc-d2/data/Earth.DB");
    long int *initial;
    while(TorO == 0)
        Get_action(0, initial, 0);
    Show(DB);
    CloseDataBase(DB);
};

```

BIBLIOGRAPHY

- [1] Crain, I. K., MacDonald C. L., "From Land inventory to Land management: the evolution of an operational GIS", *Cartographic*, 21, 1985, pp. 40-46.
- [2] Dozier, J. "Access to Data in NASA's Earth Observing System (Keynote Address). Proceeding of the International Conference of the ACM SIGMOD, page 1, San Diego, CA., June 1992.
- [3] Short, N. et. al. "Constraint Based Scheduling for the Goddard Space Flight Center Distributed Active Archive Center's Data Archive and Distribution System. NASA white paper, 1995.
- [4] Rische, N., Sun, W., Barton, D., Deng, Y., Orji, C., Alexopoulos, M., Loureiro, L., Ordonez, C., Sanchez, M., Shaposhnikov, A. "Florida International University High Performance Database Research Center." *SIGMOD Record*, 24, 3, 1995, pp. 71-76.
- [5] Rische, N. "A Database Design: The Semantic Modeling Approach. McGraw-Hill, 1992, pp. 528.
- [6] <http://eratos.erin.utoronto.ca/res/vbr/web/semant.html>
- [7] Jackson, M. J., Jame, W. J., Steven, A., "The design of Environmental Geographic Information System", *Philosophical Transactions of the Royal Society of London*, Vol. 324, 1988, pp. 77-84.
- [8] Elmasri, R., Navathe, S. B., "Fundamentals of Database Systems", The Benjamin/Cummins Publishing Company, 1994.
- [9] Rische, N., Li, Q., "Storage of Spatial Data in Semantic Database", *Computer in Engineering* 1994, (Preceding of the 1994 ASME International Computers in Engineering Conference, Minneapolis, MN, September 11-14, 1994), ASME, pp. 793-800.
- [10] Rische, N., "A file structure for Semantic Databases", *Information System*, 16, 4 (1991) pp. 375-385.
- [11] Rische, N., "Database Design Fundamentals: A Structured Introduction to Databases and a Structured Database Design Methodology. Prentice-Hall, Englewood Cliffs, NJ, 1988, pp. 436.
- [12] Bracchi, G., Paolini, P., Pelagatti, G., "Binary Logical Associations in Data Modeling", In G.M. Nijssen(ed.), *Modeling in Database Management Systems*. IFID Working Conference on Modeling in DBMSs 1976.

- [13] Frank, A. V. "Requirements for a Database Management System for a GIS", Photogrammetric Engineering and Remote Sensing, Vol. 54, No. 11, 1988, pp. 1557-1564.
- [14] Keating T., Phillips, W., Ingrm, K., "An Integrated Topologic Database Design for Geographic Information Systems", Photogrammetric Engineering and Remote Sensing, Vol. 53, No. 10, 1987, pp. 1399-1402.
- [15] Rishe, N., Accepted grant proposal to NASA. Grant NAGW-4080. 1992.
- [16] Waugh, T. C., Healey, R. G., "The GEOVIEW design: A relational database approach to geographical data handling", International Journal of Geographical Information Systems, Vol.1 No. 2, 1987, pp. 101-118.
- [17] Asker, J. R., "US/Russian satellite maps: worst Antartic Ozone Hole", Aviation Week and Space Technology, Vol. 139, No. 17, 1993.
- [18] ftp://rtfm.mit.edu/pub/usenet/news.answers/ozone_depletion/intro
- [19] Webster, C., "Desegregated GIS architecture: Lessons from recent developments in multi-site database management systems", International Journal of Geographical Information Systems, vol. 2, no. 1, 1988.
- [20] Muffin, G., "Raster versus Vector Data Encoding and Handling: A Commentary", Photogrammetric Engineering and Remote Sensing, Vol. 53, No. 10, 1987, pp. 1397-1398.
- [21] Gregg, W. et. al. SeaWiFS Technical Report Series. NASA Technical Memorandum 104566, Vol. 15. January 1994.
- [22] Rishe N., Shaposhnikov A., Sun, W., "Load Balancing Policy in Massively Parallel Semantic Databases", Proceedings of the First International Conference on Massively Parallel Computing Systems, IEEE Computer Society Press, 1994, pp. 328-333.
- [23] <http://www.usgs.gov/research/gis/application7.html>