Software for Site Specific Propagation Prediction

by

Subramanian T. Parameswaran

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of
Master of Science
in
Electrical Engineering

APPROVED

Theodore S. Rappaport, chairman

A. Lynn Abbott

Peter M. Athanas

Prabhakar M. Koushik

December 2, 1994
Blacksburg, Virginia

c.2

LD
5655
V855
1994
P373
c.2

Software for Site Specific Propagation Prediction

by

Subramanian T. Parameswaran

Theodore S. Rappaport, chairman

Electrical Engineering

(ABSTRACT)

The design of cellular wireless communication systems is influenced by the propagation characteristics of the channel and their effect on transmitted signals. The multipath propagation characteristics of a channel, except under simple transmission conditions, limits the design of emerging cellular systems. Once the propagation conditions are understood, systems can be designed more efficiently in terms of site layout and frequency planning, to reduce deployment costs and improve performance.

A software tool for site specific radio propagation prediction is being developed at the Mobile and Portable Radio Group, Virginia Tech. The objective of this research is to develop re-usable program modules that can be utilized to implement several functional features of the software.

This thesis describes the raster and vector programming libraries that have been developed during this research. These libraries are utilized in the database module, user interface and graphics module and propagation prediction module of the software. Significant among the capabilities that these libraries provide are the ability to store and retrieve raster and vector data from the database, import measured and predicted data available in standard data formats into the database, obtain user inputs through an interactive mechanism, display images of site specific information at various resolutions, and enable the user to view and analyze measured and predicted data.

This thesis also presents the data format, that has been defined during this research,

to acquire and import contour and elevation information of buildings. The SISP General Data Format (GDF), that has been developed, to facilitate import of measured and predicted data has also been described. The Hierarchical Data Format (HDF), developed by the National Center for Supercomputing Applications (NCSA), has been adopted to store measured and predicted data in the database. The programming techniques used to retrieve and display data stored in this format are also outlined in this thesis.

SISP has been developed on Sun Sparc workstations. Details of the function libraries that have been developed, programming methodologies employed, and data formats that have been defined for site specific and measured and predicted RF data are included in this thesis.

Acknowledgments

My sincere appreciation goes to my academic advisor, Dr. Theodore S. Rappaport for giving me the opportunity to work with the Mobile and Portable Radio Research Group. His advice and guidance have proved to be an invaluable part of the research experience. I also want to thank my committee members Dr. A. Lynn Abbott, and Dr. Peter M Athanas for their helpful comments and suggestions.

I am very grateful to Mr. Prabhakar M. Koushik, for providing invaluable technical guidance and motivation. Working with him on this project has been a positively enriching experience. I also thank Joe Liberti and Carl Dietrich for the thought provoking discussions and all the students for their friendship and suggestions. I am obliged to the staff at the Mobile and Portable Radio Research Group for their assistance throughout the course of this research.

I am indebted to the Advanced Research Projects Agency and the Mobile and Portable Radio Research Group for their financial support.

My deepest appreciation goes to my family and friends for their immeasurable support and encouragement. All this would not have been possible without the love and inspiring words of my grandfather Mr. K.S. Subramanian, and my mother Mrs. K.S. Bhagirathi. It is to them that I owe everything.

Table of Contents

1 Introduction 1

1.1 Wireless Communications - An Overview 1

1.2 Cellular Systems - General Aspects 4

1.3 The Significance of Propagation Prediction 4

1.4 Outline of the Thesis 6

2 Specifications for Site Specific Propagation Prediction Software 16

2.1 Overview 16

2.2 Introduction to Propagation Prediction Models 16

2.2.1 Empirical Propagation Models 17

2.2.2 Site Specific Propagation Models 18

2.3 Need for Site Specific Propagation Prediction Software 19

2.4 Functional Specifications for SISP 20

2.5 Overview Of Existing Software 26

2.5.1 MCS v1.2 26

2.5.2 MSRFM 27

2.5.3 CellCAD 27

3 SISP Raster ToolKit 29

3.1 Overview 29

3.2 Introduction to Geographical Information Systems 29

3.3 GRASS - Geographical Resources Analysis Support System 32

3.3.1 Organization of GRASS 32

3.3.2 GRASS process model 33

3.3.3 Capabilities of GRASS 34

3.3.4 GIS Features Required in SISP 34

3.3.5 Modifications to GRASS programs 35

3.4 SISP raster toolkit - An Introduction 36

3.5 SISP Raster Data Format 37
3.5.1 Cell Data File 37
3.5.2 Header file 40
3.5.3 Color file 42
3.5.4 Range file 43
3.6 The UTM Coordinate System 44
3.7 The RGB color model 47
3.8 Initialize display window 48
3.9 Displaying Raster Data 52
3.10 Displaying the legend 59
3.11 Conversions between coordinate systems 60
3.12 Continuous coordinate tracking 63
3.13 Continuous attribute display 64
3.14 Map Scale - Definition and Display 64
3.15 Overlays on displayed raster images 66
3.16 Creating icons for antenna types 69
3.17 Zoom Operation 70
3.18 Saving raster images 72
3.19 Setting color choices 72

4 SISP Vector Library 74
4.1 Overview 74
4.2 SISP Vector Data Format 74
4.2.1 Vector Data File 75
4.2.2 Vector Attribute File 80
4.3 Vector Topology Rules 81
4.4 Displaying Vector Data 82
4.5 Acquisition Of Building Data 86
4.6 AutoCAD DXF File Format 87
4.7 Conversion From DXF to SISP Binary Vector Format 91

4.8 SISP Plotting Library 96

5 **SISP Data Formats** 99

5.1 Overview 99

5.2 Specifications For Data Formats 99

5.3 Data Formats - An introduction 100

5.4 Support Types for Measured and Predicted Data 102

5.5 SISP General Data Format 102

5.5.1 The Prologue 107

5.5.2 The Data Section 108

5.6 Hierarchical Data Format (HDF) - An Introduction 110

5.7 Basic Structure of HDF Files 111

5.7.1 File Header 111

5.7.2 Data Object 111

5.7.2.1 Data Descriptor (DD) 112

5.7.2.2 Data Descriptor Blocks 114

5.7.2.3 Data Element 115

5.7.3 Sets 115

5.7.4 Groups 115

5.8 Overview of the HDF interfaces. 116

5.8.1 The Single File Scientific Data Interface (DFSD) 118

5.8.2 The File Annotations Interface (DFAN) 120

5.8.3 The Multifile Scientific Data Set Interface (SD) 121

5.9 Conversion from GDF To HDF 124

5.9.1 Version 1.0 124

5.10 Version 2.0 128

6 **Conclusions and Future Work** 134

6.1 Summary of Work 134

6.2 Future Work 135

7 References **137**

APPENDIX A: AutoCAD DXF Interface **143**

A.1.1 Introduction 143

A.1.2 DXF Files 143

A.1.3 Overview of the File Structure 144

A.1.4 DXF File Sections 146

A.1.4.1 HEADER section 146

A.1.4.2 TABLES section 146

A.1.4.3 BLOCKS section 147

A.1.4.4 ENTITIES section 147

A.1.4.5 POLYLINES entity 149

A.1.4.6 VERTEX entity 150

A.1.4.7 SEQEND 151

A.1.5 Writing DXF Interface Programs 151

APPENDIX B: Function Reference **152**

List of Figures

Figure 1-1. SISP opening screen 8
Figure 1-2. Directory setup menu 9
Figure 1-3. Display of terrain elevation data (raster data) 10
Figure 1-4. Display of terrain overlaid with building data (vector data) 11
Figure 1-5. Display of terrain, buildings, and positions of transmitters and receivers....12
Figure 1-6. Display of the E-plane pattern of an antenna 13
Figure 1-7. Display of measurement data 14
Figure 2-1. The function framework of SISP 23
Figure 3-1. Overview of a typical geographical information system 30
Figure 3-2. The GRASS process model 33
Figure 3-3. Format of an uncompressed raster data file 38
Figure 3-4. Illustration of the run-length encoding scheme 39
Figure 3-5. Format of a compressed data file 40
Figure 3-6. Format of the cell header file 41
Figure 3-7. Format of the color file 42
Figure 3-8. Alternate format of the color file 43
Figure 3-9. UTM Zones in the contiguous United States 45
Figure 3-10. A typical UTM zone 46
Figure 3-11. RGB color cube 48
Figure 3-12. Template of the Colors structure 54
Figure 3-13. Format of a GRASS-4.0 color file 55
Figure 3-14. Data structure that is used to store the color encoding scheme. 58
Figure 3-15. Color coding of raster data. 59
Figure 3-16. Solution to the UTM zone straddling problem 63
Figure 3-17. Map scale in SISP 65
Figure 3-18. Display of raster data overlaid with vector data 68
Figure 3-19. X bitmap file format 70
Figure 4-1. Outline of the organization of the SISP vector data file 76
Figure 4-2. Segment of a SISP ASCII vector data file 77
Figure 4-3. A segment of the data section of the vector data file 80
Figure 4-4. Attribute file format 81
Figure 4-5. Conceptual graphics pipeline model for vector data processing 86
Figure 4-6. Overview of the structure of DXF files 89
Figure 4-7. Segment of an AutoCAD DXF file 90
Figure 4-8. Description of a polyline entity in DXF files 93
Figure 4-9. Description of a vertex entity 95
Figure 4-10. Conceptual pipeline for the plotting library routines 97
Figure 5-1. Organization of the SISP General Data Format file 103
Figure 5-2. Examples of data descriptors and data elements 112
Figure 5-3. Fields composing a data descriptor 114
Figure 5-4. Sample data descriptor blocks 114

Figure 5-5. HDF interfaces and the organization into layers 117
Figure 5-6. Mapping GDF file contents to HDF data objects - version 1.0 125
Figure 5-7. Flowgraph of HDF functions calls - version 1.0 127
Figure 5-8. Mapping contents of GDF files to HDF files-version 2.0 130
Figure 5-9. Flowgraph of HDF function calls - version 2.0 131
Figure B-1. General format of the reference pages 152

List of Tables

Table 3-1. Database directories for raster data files 37
Table 3-2. Color quantization table with 6 levels 56
Table 4-1. Database directories for vector data....75
Table 4-2. Contents of the header section in the SISP vector data file 78
Table 5-1. The SISP General Data Format....104
Table 5-2. Fields in a data descriptor 112
Table 5-3. Functions available in the DFSD interface 119
Table 5-4. Functions available in the DFAN interface....121
Table 5-5. Functions available in the SDinterface 122
Table A.1-1. Group codes commonly present in DXF files....145
Table A.1-2. Common entity types 148
Table A.1-3. Group codes for *POLYLINE* entities and their description....149
Table A.1-4. Group 70 bit codes for the *POLYLINE* entity....149
Table A.1-5. Group codes for the *VERTEX* entity and their description 150
Table A.1-6. *VERTEX* flag codes and their description 150

Chapter 1. Introduction

1.1 Wireless Communications - An Overview

A phenomenal growth in wireless communications has been witnessed in the past few years. The demand for mobile radio services has significantly increased since their introduction in the mid 1980's. At the beginning of 1993 there were more than 10 million cellular phone users in the USA. A recent study by the Personal Communications Industry Association (PCIA) predicts that by the year 1998, broadband 2 GHz personal communication services (PCS) will have 8.55 million subscribers, paging services will have 36.8 million subscribers, and cellular services will have 33.07 million subscribers [Trw94]. The anticipated growth requires a more efficient use of the RF spectrum. Additionally, cellular subscribers expect higher quality services at economical rates [Rap91]. Cellular providers must adapt themselves to this rapidly changing environment, and utilize the existing network infrastructure efficiently.

The cellular concept was introduced by Bell Laboratories and studied extensively during the 70's [Mac79]. Prior to cellular technology, wireless systems consisted of a fixed number of channels used for transmitting and receiving information over a given geographic service area. The size of the service area was limited by the range of the base transmitting antenna. The overall capacity of the system was limited by the number of radio channels that were allocated to it.

Rather than increasing the transmitted power, cellular systems are based on the concept of frequency re-use, i.e., the same frequency is used by several sites which are located far away from each other. This keeps the co-channel interference below acceptable levels. The system capacity can be increased by decreasing the size of the cells, thereby increasing the number of times a given frequency can be re-used [Gre92].

In the US the first cellular system, the Advanced Mobile Phone Service (AMPS), was introduced in 1983 [You79]. This system is capable of tracking mobile units as they move from cell to cell. This means that calls are delivered to a unit wherever it is located,

and the connections are maintained with it as it moves within and between cells. "Hand-off" is the term used to describe how cellular systems transfer calls between channels in different sites. This first generation cellular system is based on analog narrow band frequency modulation (NBFM) around 900 MHz. Second generation cellular systems, based on digital technology and advanced networking principles, are being deployed throughout the world. These systems use either time division multiple access (TDMA) or code division multiple access (CDMA) to provide improved quality and increased capacity as compared to the analog systems [Cal88].

Forthcoming wireless systems such as personal communication systems, will provide individuals with ubiquitous voice and data services. The Federal Communications Commission (FCC) has defined Personal Communication Services (PCS) as "a family of mobile and portable radio communications services which can provide services to individuals and businesses and be integrated with a variety of competing networks" [Kob93]. The implication of this definition is that it allows PCS licensees to provide virtually every conceivable service, short of conventional radio and television broadcasting or non-mobile services.

Personal Communication Networks (PCN) is a very important member of the family of services provided by PCS. These networks will utilize sophisticated hand-held devices that will work anywhere in the world. In its full realization, PCN requires the support of a large infrastructure of indoor and outdoor base stations, subscriber location and service-preference databases, and spectrum efficient radio technologies capable of providing a high-volume service.

Several services compete for a limited amount of the electromagnetic spectrum. The North American cellular systems have 50 MHz of the spectrum allocated to them. The 12 VHF channels television channels require 72 MHz of the spectrum. The NCR WaveLAN system uses 12 MHz of the spectrum and the current generation of cordless telephones operate in approximately 1MHz of the spectrum [Cha93]. The lack of available spectrum is a serious impediment to the development and introduction of new services.

This issue is further compounded by differing wireless communication standards, artifacts of the strategies of various governments around the world and intense competition. In Europe, the Nordic Telephone system (NMT450) was developed in 1981 for the 450 MHz band. This system is presently being deployed in many countries, including Russia. Cordless telephone technologies, such as CT-2 and CT-2+, are being considered by many countries as a low-cost alternative to PCS [Lar94], [Swa90]. The Digital European Cordless Telephone (DECT) is also a Pan-European standard for cordless telephony [Och89]. The Group Special Mobile (GSM) is a Pan-European digital cellular system offering both voice and data services [Mou92]. An extension of this system is the DCS 1800, which is expected to provide the next generation PCS services [Cal88], [Tia90]. In North America the Cellular Telecommunications Industry Association (CTIA) members decided to adopt the IS-54 standard [Cox88]. The Japanese Digital Cellular (JDC) air interface standard is closer to the United States IS-54 TDMA standard than to other digital cellular standards [Cox88], [Kuw85], [Rai91].

Regulatory solutions complement spectrum-efficient technologies to alleviate this problem. Microwave radio spectrum is being allocated by regulatory bodies, like the FCC, to new, high-capacity, digital telephone and computer networks. Still, many technical and policy hurdles have to be resolved. Key among them are decisions about which parties will be allowed to hold the government granted licenses for new spectrum. Other considerations are in reallocating the spectrum, hitherto used for point-to-point use, for mobile and portable use.

Moreover, the FPLMTS (Future Public Land Mobile Telecommunication System) standard body is an international group of connectivity standards, that is considering how the PCS networks, in various countries, can be linked to form an universal personal telecommunications system. This system includes the functions of cordless telephony, paging, and private branch exchanges, and is expected to support and extend existing and future generations of cellular mobile systems [War92].

Rapid progress in digital wireless technology promises to make voice, data, and

video services available anytime, anyplace. Progress in technology, along with establishment of standards and regulatory rule making might make the "wireless vision" a reality within the next decade.

1.2 Cellular Systems - General Aspects

Cellular radio systems do not represent a completely new technology, but a stage in the process of logical development, to handle problems faced in radio system design. The traditional problem in mobile radio system design has been to handle conflicting requirements in system capacity and area coverage. A proven solution is to divide the geographic area to be covered into smaller segments called "cells", and to re-use frequency throughout the area. Frequency re-use is therefore the fundamental objective of cellular radio systems. For system planning, the given geographical area is divided into cells in a regular fashion. Each cell is represented by a hexagon and each hexagon is chosen such that the cells completely cover the desired area.

Implementation of such systems requires a careful engineering approach [Rap91a]. A radio transmission channel tends to degrade the transmitted signal. The success of these systems depends on keeping mutual interference levels between the cells to a minimum. Moreover, if the mobiles need to communicate to their base or to other terminals, all of the transmitters involved must be capable of providing enough signal strength to cover the area. Future PCS will likely use low power and low antenna heights to limit the propagation outside the cells. In these systems, it will be essential that the propagation characteristics be known in advance. Hence, it is important to have fast and accurate propagation prediction techniques.

1.3 The Significance of Propagation Prediction

A major challenge in the design of cellular systems is to overcome the effects of the varying radio link. The characteristics of the radio channel, which exists between the transmitter and receiver, are complex and heavily dependent on the nature of the physical

environment in which the equipment operates. The radio propagation between adjacent cells must be kept to a minimum so as to reduce the interference signal. It is crucial that, for the efficient use of the allocated frequency band, the designer should be able to predict the minimum power to be transmitted from a base station. It is essential that in these systems, the propagation characteristics of the channel be known ahead of time. This will help in determining the power to be transmitted from the base station in order to provide the optimum coverage for a given cell.

Wireline systems use transmission lines to connect communication terminals. The effect these transmission lines have on a signal are well known [Lia85]. Unlike in physical media (such as copper or fiber) which have a controlled transmission environment, in a mobile channel, the impulse response of the channel is constantly changing. Thus system design for mobile telephony is a considerably complex task. The propagation models aim to characterize the channel accurately, by considering several site specific phenomena which are likely to affect the signals transmitted in this environment.

Under simple transmission conditions, the signals from the transmitter to the receiver might travel along a single path. However, in practice, the signal can travel to the receiver via more than one path. This situation is called 'multipath'. It occurs because the signal undergoes reflection and scattering from the surfaces of buildings, and other obstacles along the path. Additionally, since the terminals are mobile, the channel from the transmitter to the receiver is constantly changing and the radio waves arrive at the receiver at different times. These are vectorially summed to give the resultant signal at the receiver [Par89]. A deep fade occurs when these signals arrive at the receiver out of phase. This type of fading is time-selective, since the fade varies with time depending on the position of the mobile unit. Fading can also be frequency-selective, in which a portion of the spectrum occupied by the signal is subject to a deep fade.

If a fixed number of radio channels are available for use in a system, they can be divided into a number of sets, each set being allocated for a small area, served by a single base station. Since the area over which the mobile units transmit is small, the frequencies

can be re-used. Radio signals could be interfered with by other systems nearby using the same frequencies. This phenomenon is called co-channel interference. The propagation models help to estimate the degradation in system performance, due to co-channel interference, resulting from the frequency re-use techniques.

Accurate propagation models can allow researchers to determine the appropriate modulation, equalization, and multiple access techniques that are most suited for wireless systems in multipath propagation environments. System throughput and acceptable bit error rate will likely be application specific, and the appropriate propagation models will be needed to efficiently design these systems.

1.4 Outline of the Thesis

This thesis describes the implementation of re-usable program libraries, utilized in the development of a software tool for site specific propagation prediction and wireless communication system design--SISP. SISP is being developed by graduate students and researchers at the Mobile and Portable Radio Group.

Chapter 1 of this thesis provides an overview of wireless communications, and the significance of propagation prediction. The various propagation models that currently exist are discussed in Chapter 2. This discussion serves as an overview of the models, and introduces the resources that the computer models for propagation prediction utilize. The need for site specific propagation prediction software is emphasized, and the functional specifications for SISP, that have been established to serve as guidelines for the work described in this thesis, are also discussed in this chapter. Chapter 2 also includes a survey of the tools for site specific propagation prediction and wireless communications system design, that currently exist.

Chapters 3 and 4 discuss the raster and vector programming libraries respectively, that have been developed as re-usable program modules to implement several functional features of SISP. Chapter 3 includes a brief introduction to geographical information systems in general, and the Geographical Resources Analysis Support System (GRASS) in

particular. The functional similarities that exist between GISs and site specific propagation prediction software are highlighted. The process model of GRASS is compared to the process model of SISP, and the modifications that were made to GRASS programs incorporated into the raster library have been presented. This chapter also details the functional capabilities of the raster programming library.

Chapter 4 discusses the vector programming library that has been developed for the visualization of vector data. The difficulties in handling the diverse data formats in which building contour and elevation data exist are presented; a standard building data format for SISP has been defined to overcome this problem. A detailed description of the format and the filters that have been implemented to import building data files into the SISP database are also included in this chapter. Additionally, this chapter discusses the capabilities of the plotting library that has been developed to display antenna patterns and RF measured and predicted data.

Data formats for measured and predicted data are discussed in Chapter 5. This chapter briefly describes the SISP General Data Format (GDF), developed at the Mobile and Portable Radio Group (MPRG), for measured and predicted data. This format serves as a standard, user-readable, ASCII format for data to be imported into the SISP database. Since rapid access and retrieval of data from the database is critical for analysis and display operations, a indexed binary format is utilized for storing measured and predicted data in the SISP database. The Hierarchical Data Format (HDF), developed by the National Center for Supercomputing Applications (NCSA), has been adopted for this purpose. A brief overview of this format and its programming interface is provided in this chapter. Description of programs that have been developed to store and retrieve data in HDF is also presented in Chapter 5.

This chapter has described the growth and the emerging scenario in wireless communications. The importance of accurate and computationally efficient propagation models and the need for software tools that aid research and commercial system design was emphasized. Sample outputs from SISP are shown in Figures 1-1 through 1-7.

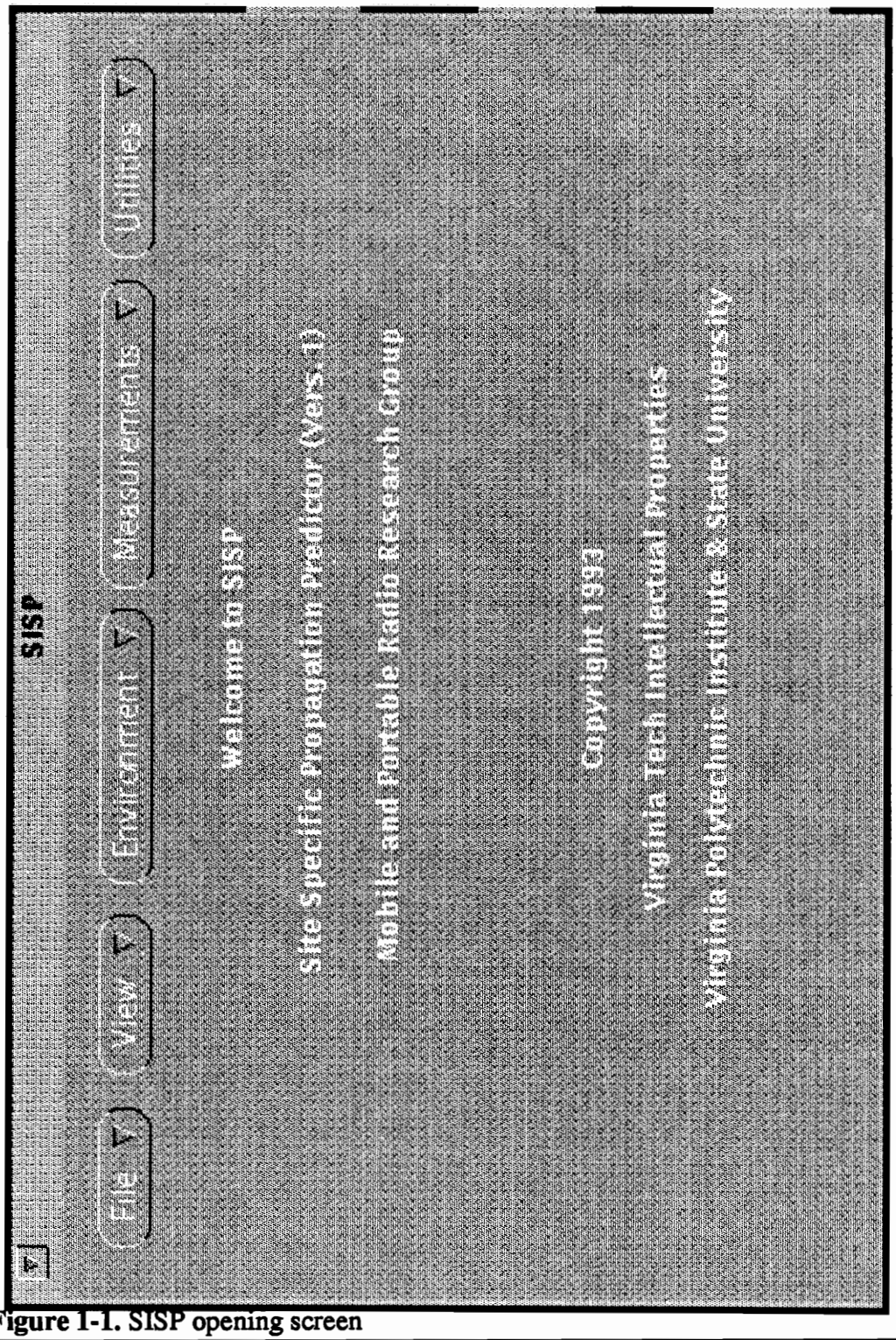

Figure 1-1. SISP opening screen

Figure 1-2. Directory setup menu

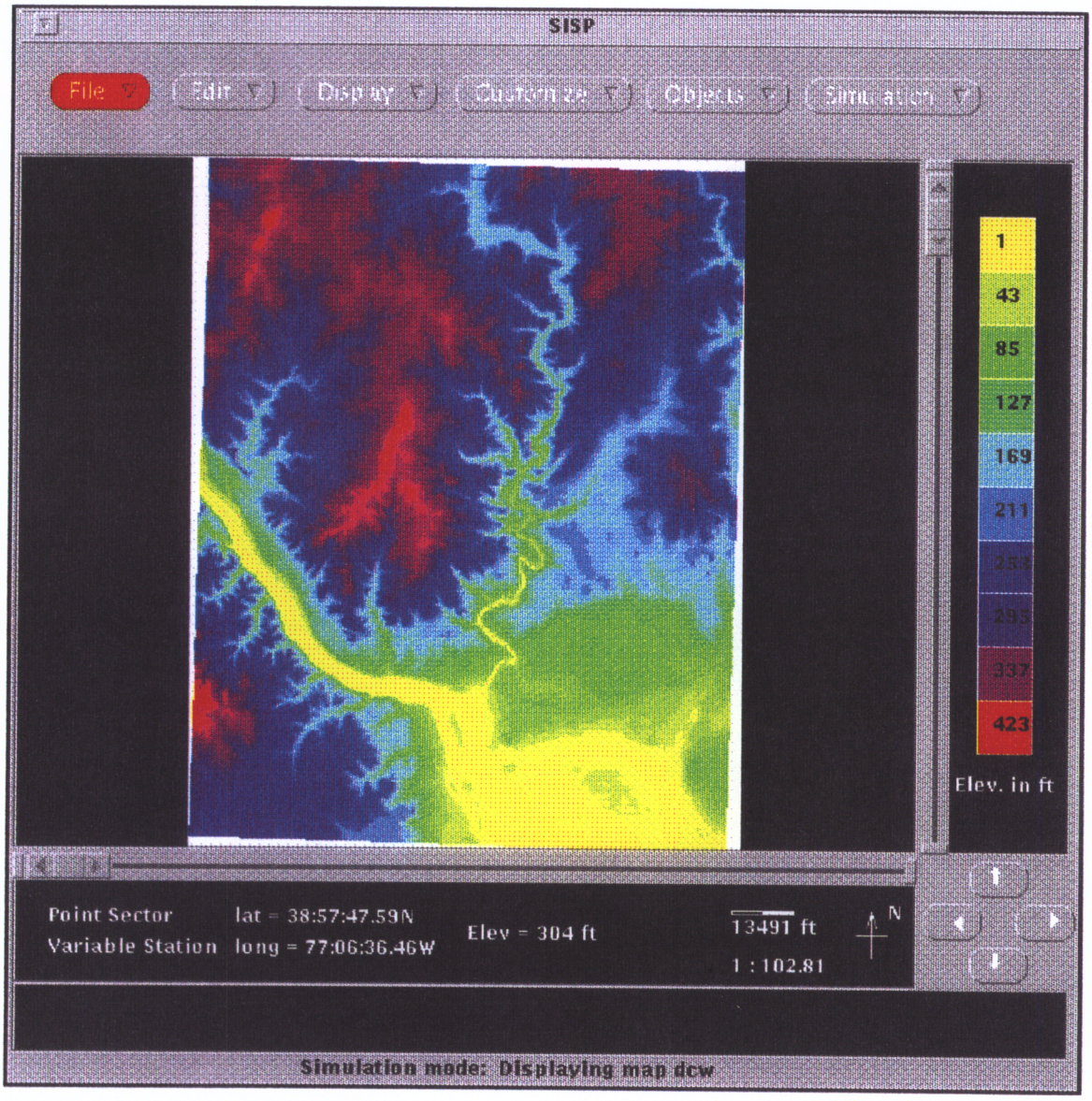

Figure 1-3. Display of terrain elevation data (raster data).

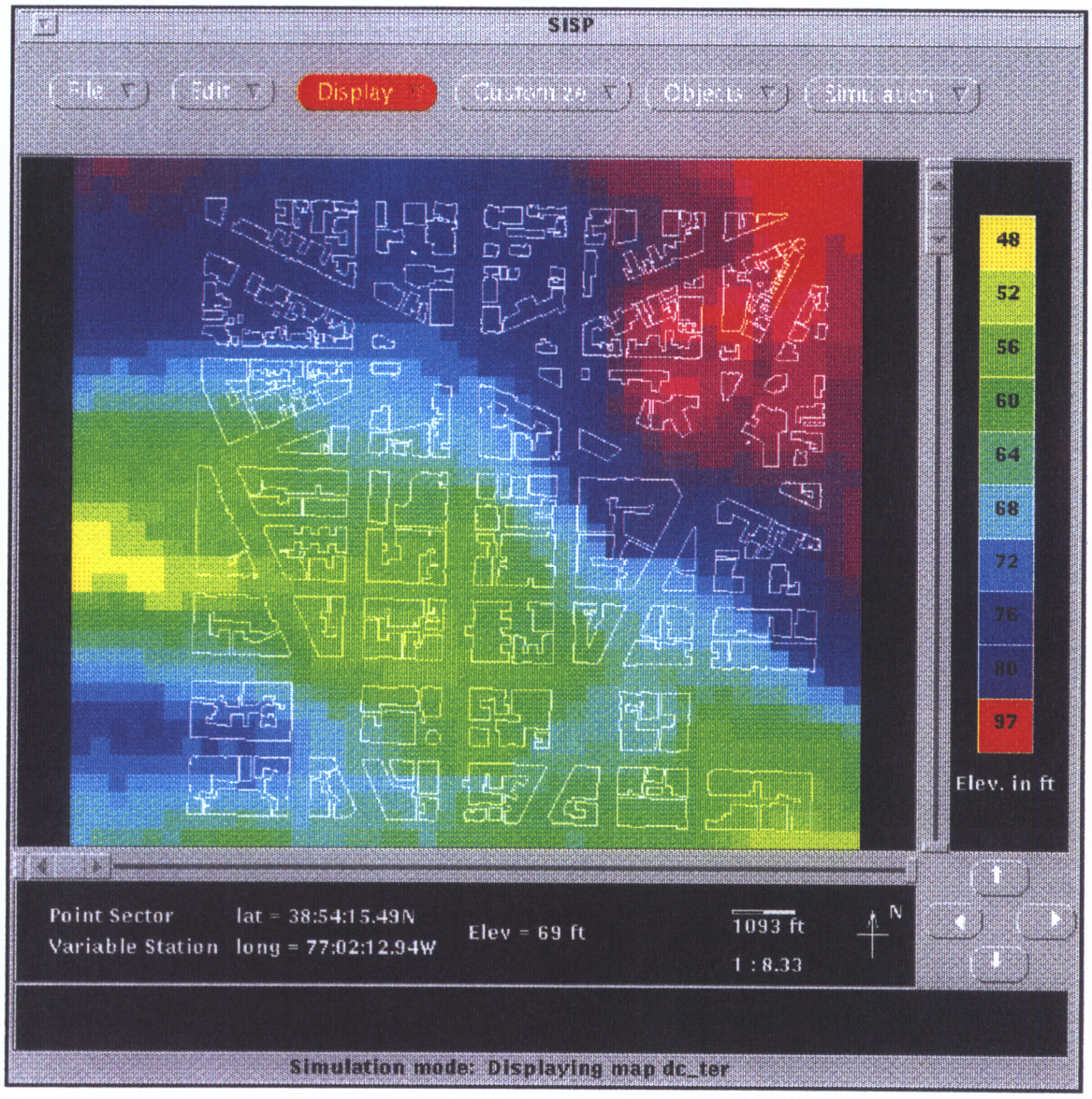

Figure 1-4. Display of terrain overlaid with building data (vector data).

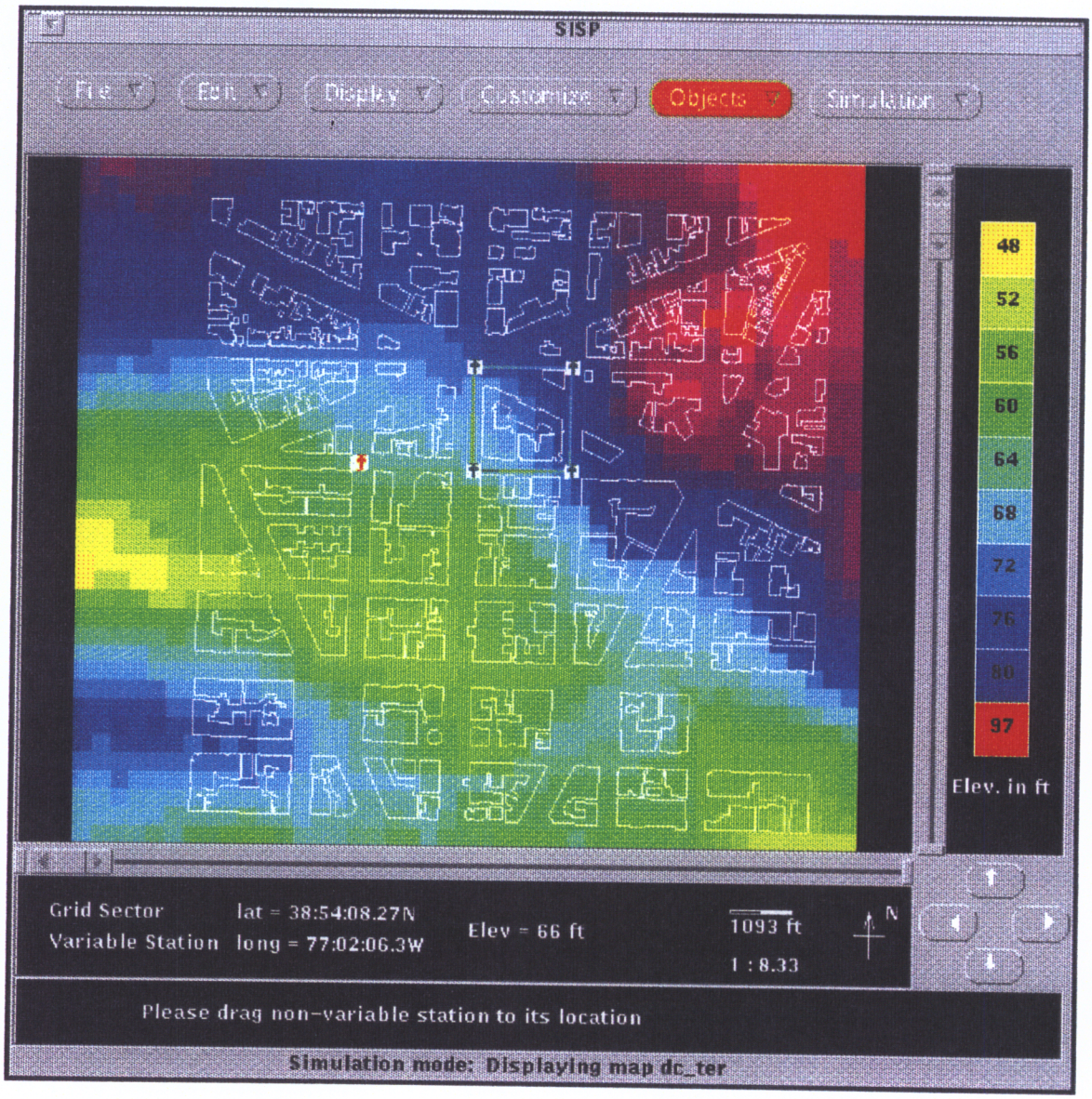

Figure 1-5. Display of terrain, buildings, and positions of transmitters and receivers.

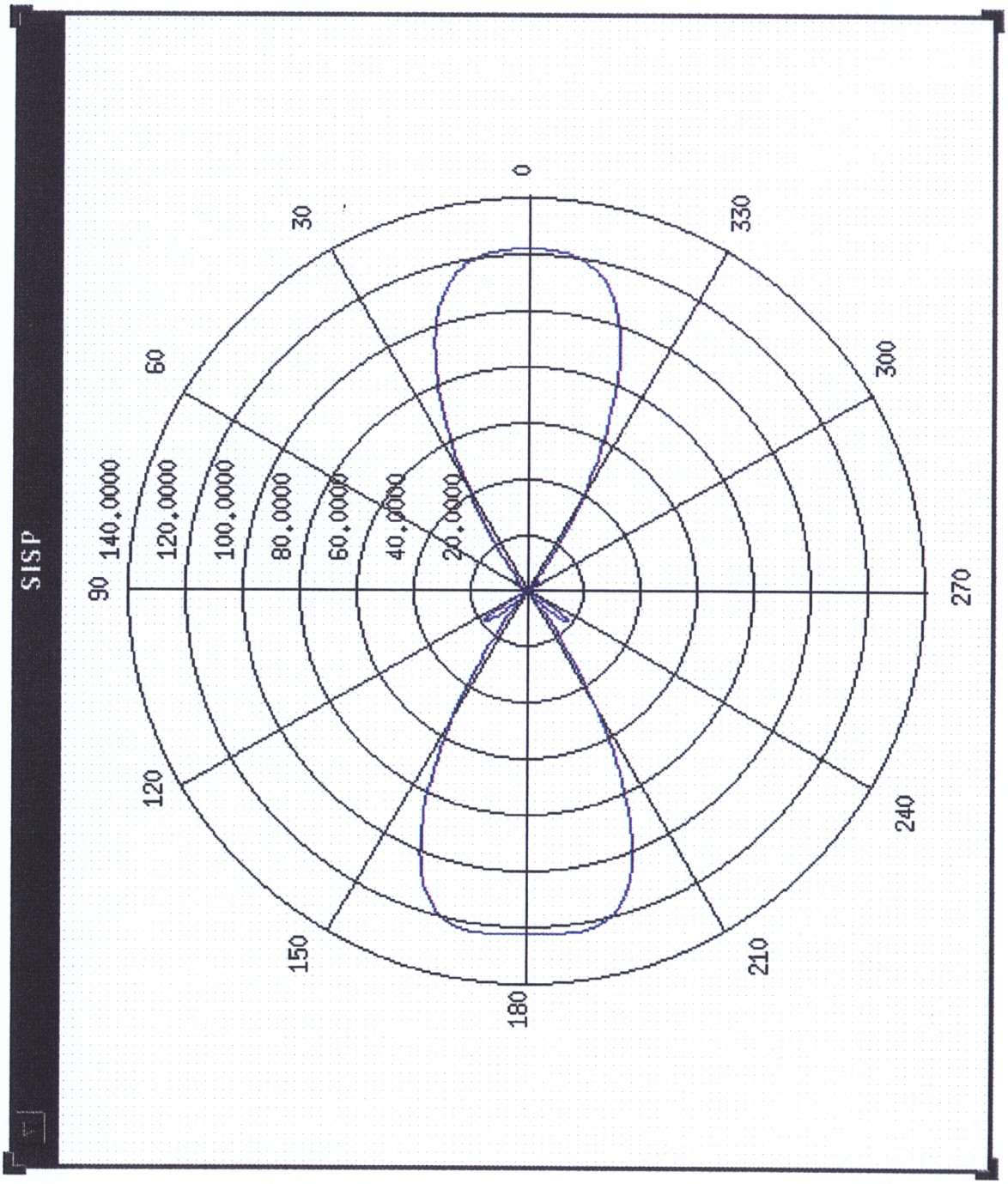

Figure 1-6. Display of the E-plane pattern of an antenna.

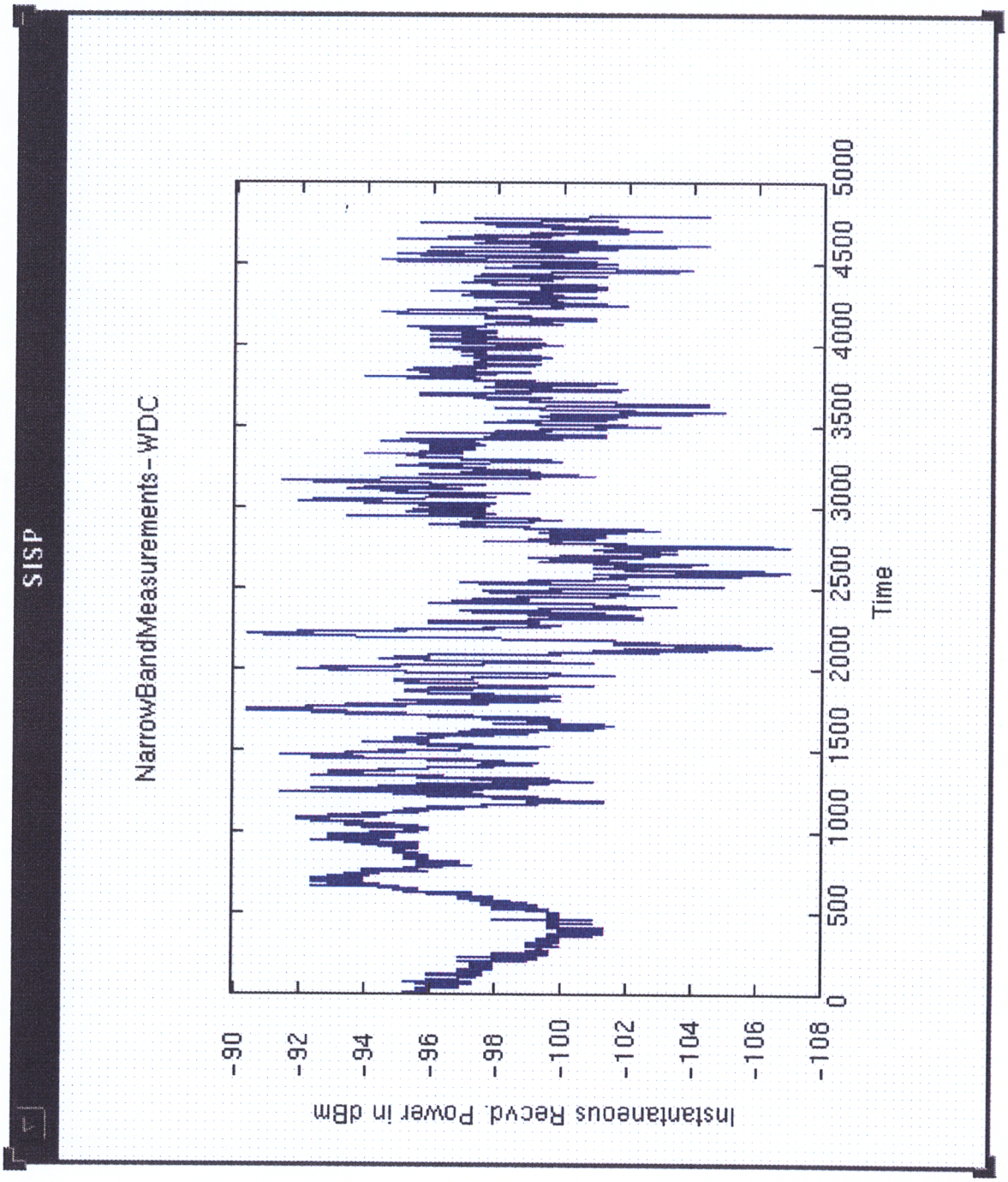

Figure 1-7. Display of measurement data.

Figure 1-1 shows the opening screen for the software. The directories which compose the database can be specified through the directory setup menu shown in Figure 1-2. The terrain elevation data obtained from the United States Geological Survey (USGS) for Washington D.C. is shown in Figure 1-3, and Figure 1-4 displays the terrain overlaid with building contour information for the same area. Figure 1-5 illustrates the feature of the software that allows positioning of transmitters and receivers interactively. Figure 1-6 and Figure 1-7 display the outputs of the plotting library. The E-plane pattern of an antenna is shown in Figure 1-6, and data obtained from narrow band measurements is shown in Figure 1-7.

The rest of this thesis presents the work that has been done in developing a software tool for propagation prediction, and the various considerations that were involved in its development. The software tool has the flexibility and potential to complement research efforts in propagation prediction, and to serve as a commercial tool for wireless system design and support.

Chapter 2. Specifications for Site Specific Propagation Prediction Software

2.1 Overview

The radio propagation environment exerts a profound influence on the performance of wireless communication systems. In a mobile environment, the communications terminal is free to roam throughout a given geographical area. The propagation characteristics of the radio link thus vary rapidly. The radio communication links are subjected to conditions in which the energy can travel from the transmitter to the receiver by more than one path. Historically, field measurements have been used to determine the parameters of a radio system that will provide efficient and reliable coverage over a specified service area. This technique has the disadvantage of being time consuming and very expensive. Thus, accurate computer modeling of radio propagation is important, so that designers may, using a computer, examine the various trade-offs necessary in the design of wireless systems.

This chapter provides a brief overview of some of the propagation prediction models that have been developed to date. It establishes the need for having a robust software package for site specific propagation prediction, and outlines the features and functional specifications of any such propagation prediction tool. Following this, the chapter provides an introduction to the software development effort to create the programming framework for SISP, which is also the significant contribution of this thesis. An overview of some of the commercially available software is also presented in this chapter.

2.2 Introduction to Propagation Prediction Models

Mobile radio channels are complex and strongly influenced by the nature of the environment. Hence, they do not lend themselves to easy analysis. Historically, several researchers have directed efforts at modeling and characterizing the radio environment. The primary task in these efforts has been to conduct extensive measurements, in an

attempt to characterize the radio propagation environment [Hua92], [Mel93], [Mog91], [Rap90]. Although the results of these measurements have provided valuable information about different propagation environments, these efforts are expensive. Hence, accurate propagation prediction models are required to aid system design and enhance performance in future wireless systems.

Several techniques to predict radio propagation in land mobile systems exist [Hat80], [Oku68], [Ros93], [Wal88]. Many of the earlier techniques are based on theoretical equations, describing propagation in free space or over a plane surface, and including an additional term to compensate for the diffraction losses in the environment. However, in mobile communication systems the transmission path from the transmitter to the receiver can vary from line-of-sight, to one obstructed by objects in the environment. Moreover, the mechanisms contributing to electromagnetic wave propagation depend on the nature of the obstructions in the environment. Thus these models suffer from not being accurate enough to characterize all propagation environments. These models have evolved using a combination of experimental and analytical techniques. The sections below outline some of the popular, outdoor signal strength and path loss prediction models.

2.2.1 Empirical Propagation Models

The empirical propagation models have been derived from propagation prediction studies, made in an attempt to characterize radio wave propagation for cellular communications. The empirical approach is based on the techniques of fitting curves or analytical expressions that closely approximate a set of measured data. The advantage of these models is that they implicitly account for all the factors that affect propagation. However, the disadvantage of these models is that the validity of these models, in different environments and at different operating frequencies, needs to be established by measurements made under the new conditions.

The log-distance path loss model is based on observations that the received signal power at any point decreases exponentially with the distance of separation from the transmitter. The value of the path loss exponent, which indicates the rate at which the path loss

increases with the distance, for this model is varied according to the nature of the propagation environment. In practice, the value of the path loss exponent is computed using linear regression such that the error between the measured and estimated path losses are minimized.

The Okumura-Hata model [Oku68], [Hat80], which predicts the field strength from empirical formulas is one such model. This method is essentially based on determining the free-space path loss between the transmitter and the receiver, adding loss for the urban environment, and then using correction factors to account for the nature of the terrain, the extent of urbanization, the heights of the antennas, and orientation [Par89]. The model is based entirely on measured data and does not provide any analytical explanations. The model is good for urban and suburban areas, but not as good for areas with irregular terrain.

2.2.2 Site Specific Propagation Models

The empirical models described in the previous section do not explicitly account for the various factors in the environment. Thus they are not valid for any type of propagation environment. However, the site specific propagation models [Sch91], [Rus91], [Sch94] take into account the factors in the environment that influence electromagnetic wave propagation. Thus these models utilize data about the terrain, buildings, foliage and man-made objects in the environment. The electromagnetic characteristics of these features can also be incorporated into these models. Since these models do not make any assumptions about the nature of the environment, they are valid for different environments, provided the site specific data is available.

The Russell model [Rus91], [Sch94] utilizes data available about the buildings in the environment. The data essentially contains the contours of buildings in the area and their heights. The model utilizes three dimensional multiple knife edge diffraction techniques, to generate predictions for any given area. The model searches the building database for true diffracting edges over roofs and sides of buildings to compute the predictions.

The site specific propagation modeling work of Schaubach [Sch91], [Sch94] utilizes three dimensional ray tracing techniques to generate predictions for microcell environments. The models use information from a building database, and use techniques such as three dimensional ray tracing, geometrical optics, and wedge diffraction theory to generate predicted data. Though these models utilize information about the buildings in the environment they do not account for the effect of the underlying terrain.

The Kurner-Cichon-Wiesbeck model [Kur92], for rural and urban areas, employs 2D and 3D ray tracing, using the ray-transportation matrix (RTM) algorithm, to determine coverage. This model utilizes a digital terrain and building database. The 2D model takes into account all the ray paths along the vertical Tx-Rx plane. The Uniform Geometrical Theory of Diffraction (UTD) is used for the calculation of diffraction effects. In the 3D extension, the computation of the scattering from points on the vertical plane is done based on the theory of bistatic scattering.

Although the site specific models are computationally intensive, they are accurate and can be used for any propagation environment. However, the model is limited by the availability of digitized terrain and building data.

2.3 Need for Site Specific Propagation Prediction Software

Site specific propagation models interact with voluminous terrain elevation and building databases. Moreover, they manipulate RF measurement and antenna pattern data for visualization and analysis. Since all this data is obtained from different sources and in diverse formats, it represents the biggest hurdle in interacting with the database. Currently, several different software packages are used to process this data, and this places the onus of mastering the use of several software packages, on the user. These difficulties can be eliminated with the development of a single software tool that provides all the desired functionality. Additionally, a standard format and technique for importing data into this software can be established.

The propagation models for indoor and outdoor environments utilize different

database elements. For instance, the outdoor models utilize building data which specifies the contours and the heights for the buildings in an area. However, the indoor models utilize detailed floor plans of individual buildings. By developing a common platform from which to execute the models, a seamless interface to the models and the site specific data that each utilize can be provided.

The preliminary step in the execution of the propagation models is to identify the region for which the predictions are desired. Often the terrain data is obtained for large areas, while the coverage is desired for only a sub-region of the entire area. A software package with a robust interface can permit the geographical region to be specified interactively. It can also allow, within this interactive framework, the specification of locations of transmitters and receivers in the area, viewing antenna pattern files, and providing parameters for the various propagation models.

Visualization of data generated by the propagation models provides critical feedback about the performance of the models. As stated earlier, the measured data is also imported into the database for visualization and analysis. The graphic libraries which form a part of the propagation prediction software can aid in the visual comparison of measured and predicted data. This functionality is essential for refining and optimizing the performance of the prediction models on the basis of the measurements.

The key points which emphasize the need for a comprehensive propagation prediction and system design software package has been outlined in this section. The next section discusses the functional specifications of the software package.

2.4 Functional Specifications for SISP

This section outlines the functional specifications which serve as the guidelines for the development of SISP. These specifications were devised at MPRG to shape the software to be a useful research tool [Sis94], [Rap93]. A detailed discussion of the functional specifications, classified according to their functionality, is provided in this section.

Database module

- Provide the ability to efficiently organize a database of site specific information
- Import terrain and building data, obtained in various formats, into the database
- Maintain a database of antenna pattern files
- Maintain a database building material properties
- Support an ASCII file format to serve as a standard input format for importing RF measurement data
- Support a binary data format for the internal representation of measured and predicted data
- Employ a planimetric coordinate system like the Universal Transverse Mercator (UTM) system for referencing terrain data
- Handle other coordinate systems, such as the geographical coordinate system (GCS) (latitude-longitude), and provide coordinate transformation factors to go between the different systems

User Interface and Graphics Module

- Provide an interactive framework for viewing and plotting terrain and building data at any resolution
- Allow the geographical region of interest to be specified
- Allow for interactive positioning of transmitters and receivers
- Provide the ability to view antenna pattern files from the database and associate them with the positions of transmitters and receivers
- Provide for visualization and analysis of predicted and measured data

Propagation Prediction Module

- Provide a integrated platform for the execution of indoor and outdoor propagation models, which may eventually support indoor and outdoor propagation prediction
- Enable the user to supply appropriate parameters for the propagation models
- Provide the flexibility for researchers to build new models which can easily "plug-in" to the existing software
- Integrate and analyze predicted and measured data for a wide range of data formats
- Utilize the measured data to optimize the propagation prediction models

The general framework for SISP is shown in Fig 2-1.

The software should provide the capability to create and organize a database containing all site specific information. The database should be efficiently organized to optimize disk storage and to provide streamlined access to commonly used database elements among multiple users. Typically, the database consists of terrain data, building data, data about other objects in the environment, material properties, and antenna pattern libraries.

In addition to organizing the database efficiently, the software should provide the capability to import terrain data obtained in the United States Geological Service (USGS) Digital Elevation Model (DEM) format. Often, building data is obtained as AutoCAD DXF files. The data from the AutoCAD DXF files should be imported into the SISP database.

The software should facilitate the creation and maintenance of a database of antenna pattern files containing data for the antenna pattern in the E-plane and in the H-plane. A binary format should be used for this data, in the database, to support an efficient mechanism to view the patterns.

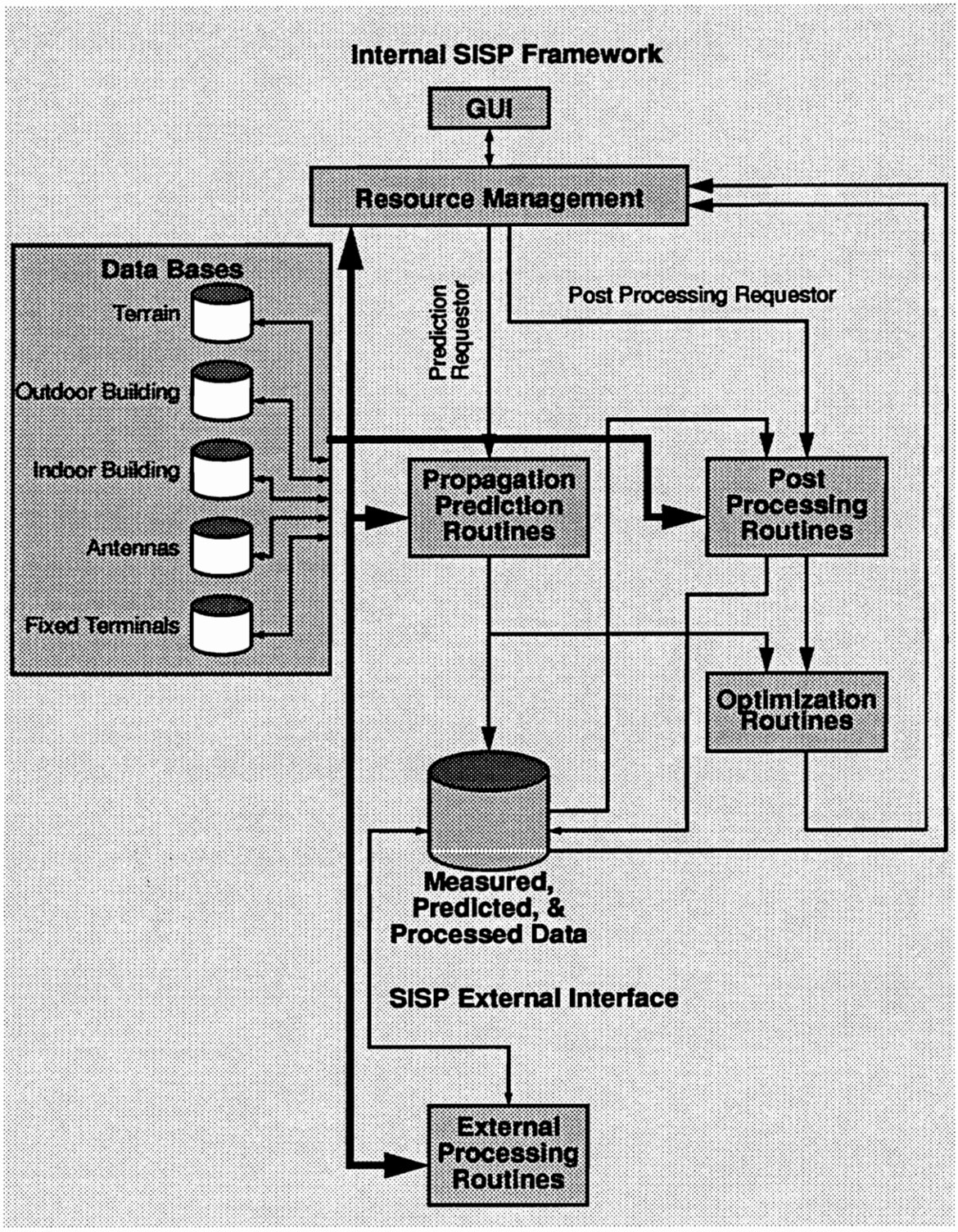

Figure 2-1. The function framework of SISP[Sis94]

The primary mechanisms in electromagnetic propagation have been identified as reflection, diffraction, and scattering. Modeling the material characteristics of buildings, which influence these mechanisms, is important. Other electromagnetic properties of materials such as conductivity, permittivity, and transmission at various frequencies should also be maintained in the database. This information should be used in conjunction with the building data. The software should be capable of importing these material characteristics into the database, when such data becomes available.

The software should be able to import RF measurement data obtained from standard measurement systems into the database. In order to aid this process, the measurement data should be converted to an open, user-readable format which serves as the input to the filters that import data. The imported data should be stored in a portable, binary format to enable the transfer of data across several computing platforms and to aid rapid retrieval for analysis and visualization. The measurement data imported into the database should be used for analysis and comparison with the predicted data. The models should be optimized by the software on the basis of this analysis.

A planimetric coordinate system, such as the Universal Transverse Mercator (UTM) system, should be adopted by the software and be used for any computations and analyses. Terrain and building data should be referenced to this system. Although a planimetric coordinate system should be the native coordinate system, the software should support other coordinate systems such as the geographical coordinate system (latitude-longitude). It should also allow for coordinate transformations between the different systems.

The software should provide a robust user interface. It should allow the terrain data for any geographical region to be displayed and overlaid with building data. The displayed image should be displayed at different zoom factors based on input from the user. The software should allow the region of interest to be specified interactively by allowing scrolling and panning operations on the displayed image.

In addition to providing the capability to identify the region of interest, the soft-

ware should allow interactive positioning of transmitters and receivers that are used by the propagation prediction routines. It should allow the various antenna parameters such as the height of the antenna, the orientation of the antenna, the transmitting frequency, the transmitted power, and the type of antenna to be specified. It should also enable the visualization of the H-plane, E-plane, and three dimensional antenna pattern.

The visualization of predicted data is important for determining the performance of the system. The software should be capable of supporting the display of data of different types and dimensionality.

- Wideband data, such as power delay profile, complex channel impulse response, angle of arrival, RMS delay spread, and local average received power
- Narrow band data, such as instantaneous received power and local average received power
- Miscellaneous data, such as the bit error rate (BER), outage probability, and carrier-to-interference ratio (C/I)
- Other user-defined data types to handle other measurement or prediction parameters

The software should serve as an integrated platform to execute program modules that implement the various radio propagation models, for both outdoor and indoor radio environments. The models can be empirical models or site specific models. The software should provide the facility to choose a propagation model to be used for predictions. As each model requires several repetitive inputs, the software should provide capability to interactively provide the model specific parameters.

The software development has been targeted for the Sun Sparc workstations but can be easily ported to other Unix based platforms.

2.5 Overview Of Existing Software

This section discusses some of the existing propagation prediction software tools. A brief overview of the tools and a description of their capabilities have been provided.

2.5.1 MCS v1.2

MCS [Mcs93] is a commercial, PC based, ray tracing propagation prediction program developed by EDX Engineering Inc., Eugene, Oregon. The software can be used for microcell, PCS and wireless LAN systems design, and channel modeling. The latest version which is a 32-bit application, is optimized for improved speed and faster memory access.

MCS is capable of handling the terrain and building databases that EDX supplies. It features ray-tracing propagation prediction models which consider various propagation mechanisms and effects. Important among them are free space, line-of-sight propagation, wall reflection, wall transmission, wall scattering, and rooftop diffraction. The propagation models combine image theory with proprietary geometric decimation of the propagating environment for very fast and efficient ray tracing. The horizontal, vertical and circular polarization propagation results are all simultaneously calculated. The models can be applied to outdoor PCS microcell environments, indoor PCS, or wireless LAN environments.

The software has adequate tools for analysis of power delay profiles, ray paths plotted on building database maps, channel time signature analysis through fading, channel spectrum signature analysis through fading, and RMS delay spread. Additionally, it aids the analysis, over a path, of received power level vs. location, RMS delay spread vs. location, most likely server vs. location, and BER vs. location. It provides for analysis of grid data such as received power level maps, RMS delay spread maps, most likely server maps, BER maps, and C/I ratio maps.

A major shortcoming of the software is that the propagation prediction models utilize simplistic terrain and building databases. Since the environment influences radio

propagation considerably, the models are less accurate than desired.

2.5.2 MSRFM

MSRFM (MultiSite RF Model) [Nov93] is a versatile radio propagation and performance modeling system developed by NovAtel Communications Ltd., Calgary, Canada. The software which executes on Unix-based workstations, has been used for the design of a large number of cellular systems.

The modeling system consists of three major functional classifications: propagation model, database utilities, and analysis programs. The software is based on the Canadian Communications Research Center's (C.R.C) VHF/UHF Propagation Prediction Program and the United States National Telecommunications and Information Administration's TIREM (Terrain Integrated Rough Earth Model). The models compute the terrain, clutter, and atmospheric losses. The effect of antenna patterns is then applied to the loss figure. Data taken from field measurements made in a wide variety of environments are used in an on-going manner to improve the accuracy of the computations.

The software can handle terrain data in a variety of formats such as DEM (Digital Elevation Model) files, USGS data files, and EMR (Dept. of Energy Mines and Resources, Canada) files. It also utilizes USGS Land Use Codes as a source of clutter data. The UTM coordinate reference system is used for referencing locations in a geographical region. The software also provides the capability to digitize maps in areas where previously digitized databases are not available.

The analysis tools provide the capability to analyze and combine MSRFM output data from multiple cell sites. Coverage prediction, best server plots, C/I ratios to show either co-channel interference or adjacent channel interference, and critical hand-off parameters can be computed by the software.

2.5.3 CellCAD

Cell CAD (Cellular Computer Aided Design) [Lcc93] is a network engineering tool for mobile radio networks developed by LCC Inc., Arlington, Virginia. It provides to

RF network planners a full range of engineering software features to create new networks and optimize the performance of existing systems. The software is developed for Unix-based hardware platforms.

The propagation models include the United States National Telecommunications and Information Administration's TIREM (Terrain Integrated Rough Earth Model) model besides other proprietary models. The models utilize highly sophisticated photogrammetry databases supplied by LCC. The models also incorporate the effect of the antenna patterns when generating results.

The software provides for the visualization of terrain data, antenna patterns, and propagation prediction results. It can be utilized for coverage analysis, channelization analysis, and interference analysis. The software provides for the integration of measured and predicted data and uses the results of the analysis to refine and enhance propagation models.

LCC also markets ANET [Lcc93], which is a PC-based network engineering tool for mobile radio networks.

The other propagation prediction and system design software in the marketplace are PlaNet [Pla94] by Mobile Systems International and Phoenix [Air93] from AirTouch Communications.

This chapter has provided an introduction to existing propagation prediction models. It emphasized the importance of site specific propagation prediction software and defined the functional specifications for SISP. This chapter also provided a survey of some of the existing software packages and highlighted their salient features. The subsequent chapters provide a detailed description of the functional aspects of SISP. They also document the programming libraries that have been developed to meet the specifications introduced in this chapter.

Chapter 3. SISP Raster ToolKit

3.1 Overview

The functional specifications for a site specific propagation prediction tool were outlined in Chapter 2. As discussed in Section 2.4, the software tool should be capable of handling large amounts of spatial data and other site specific information. The SISP raster function library is a set of programs for manipulating and displaying raster images, and forms the basis for the storage, analysis, retrieval, and display of all site specific information in raster format.

This chapter presents the background that led to the development of the raster programming library. It provides an introduction to geographical information systems in general, and a detailed description of the Geographical Resources and Analysis Support System (GRASS), in particular. The raster library is based on programs that are a part of the standard GRASS software distribution, available from the U.S. Army Corps of Engineers Construction Engineering Research Laboratory (USA-CERL) [Wes91]. The process model of GRASS is explained and the factors that necessitated extensive modifications of its programs are presented. A detailed description of the capabilities of the function library is given in this chapter. The references for some important functions in the raster library are provided in Appendix B.

3.2 Introduction to Geographical Information Systems

A geographical information system (GIS) is a tool for collecting, storing, retrieving, analyzing, and displaying data referenced by spatial or geographic coordinates, for numerous purposes [Bur86], [Sta90], [Kor92]. There are five essential elements a GIS must contain: data acquisition, preprocessing, data management, manipulation and analysis, and product generation. Figure 3-1 provides the overview of a typical geographical information system.

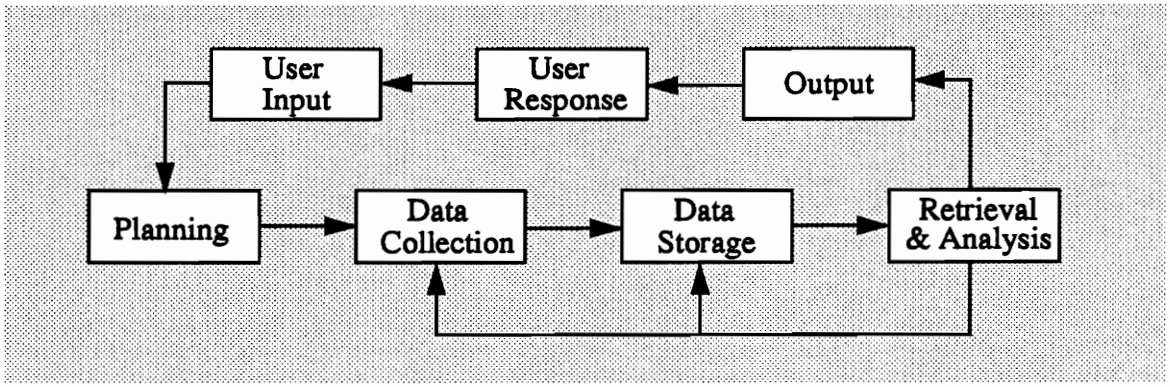

Figure 3-1. Overview of a typical geographical information system

A brief discussion of the functional elements is given below.

Data acquisition is the process of identifying and gathering the data for the application. This is an important process since the accuracy of any analytical results generated by the system depends on the precision and accuracy of the underlying data sets.

Preprocessing involves manipulating the data in several ways so that it may be entered into the database of the GIS. The principal tasks in this process are the data format conversion, and identification of the location of objects in the original data in a systematic manner.

Data management functions govern the creation of, and access to, the database itself. These functions provide consistent methods for data entry, update, deletion, and retrieval.

Manipulation and analysis is the process in which the analytical operators work with the database contents to derive new information. Additionally, the modularity that these systems provide enables other data processing and analysis systems to be linked to it.

Product generation is the phase where the final output from the GIS is created. The output products might include displayed images, hard copy maps, statistical reports,

and other graphics.

From the description of the functional elements of geographical information systems, in the preceding paragraphs, it is apparent that the capabilities of these systems are multitudinous. Besides integrating the functionality of computer aided design (CAD) and database management systems (DBMS), GISs incorporate other functional features that are outside the scope of CAD and DBMS software[Cow88]. GISs are similar to CAD systems in their capacity to manipulate objects within a frame of reference. These discrete objects can have attributes such as line thickness, line color, line type etc., assigned to each of them. In addition to handling geometrically well defined objects, GISs are also capable of handling voluminous spatial data, which is any data with spatial distribution. They also possess greater analytical capability than do CAD systems, which instead emphasize robust graphics capability.

Besides performing functions similar to CAD systems, geographical information systems also possess a number of attributes that are common to all database management systems. These include the capacity to manage large data sets, the capability to handle multiple users and databases, data independence and integrity. GISs differ from database management systems in that they provide better methods for the visualization of the data. Another difference is that database management systems employ better techniques for data storage and retrieval than do GISs, since data storage and retrieval are the principal functions that DBMSs performs.

Traditionally, geographical information systems evolved as a means of assembling and analyzing diverse spatial data. Many systems have been developed for land use mapping and planning, natural resource management, geological exploration, and transportation. Some popular GIS systems are IDRISI [Eas88], ERDAS [Erd91], ARC/INFO [Arc88], MapInfo [Map93], and GRASS [Wes91]. The versatility of these systems is apparent and they are rapidly attracting newer applications, such as the application outlined here and in [Raj92a].

3.3 GRASS - Geographical Resources Analysis Support System

GRASS is a public domain, geographical information system, designed and developed at the U.S. Army Corps of Engineers Construction Research Laboratory (USA-CERL). GRASS provides software capabilities suitable for organizing, retrieving and analyzing digital spatial data. The software is continuously maintained and updated by USA-CERL. Several other federal and commercial organizations provide distribution and support. The GRASS distribution includes an application programming interface (API) for the database management functionality and the graphics capability that the software provides.

The advantages that GRASS possesses are its portability across several hardware platforms and its availability in the public domain. As was stated earlier, the usefulness of GRASS in propagation prediction has been demonstrated by previous work in the MPRG [Raj92]. GRASS was thus used to identify the extent of GIS functionality, and provide the basic framework, needed for the development of the site specific propagation prediction software.

3.3.1 Organization of GRASS

GRASS is a collection of programs and an interface library, implemented in C, intended for programmers who desire to enhance its functionality. These programs execute under a GRASS shell which provides the environment in which these programs execute. The environmental variables used by all the GRASS programs are initialized when the environment is created. The functionality of displaying images on the screen is handled by a separate process called the graphics driver. This structure ensures that new programs can be integrated with the software without modifying existing functions. It also serves to encapsulate the system dependent display capabilities in the graphics driver. The interface library ensures that the structure of all GRASS programs are consistent, and programmers use a standard mechanism to access the database and display the information. It also reduces the time taken to develop GRASS programs that implement new applications.

3.3.2 GRASS process model

As stated previously, the GRASS system consists of two distinct entities: the programs which accomplish the storage, retrieval, and analysis of data; the graphics driver which redirects the results of the analysis to the display device. The process model of GRASS is shown in Fig 3-2.

The executing programs and the graphics driver communicate by means of named pipes, which are a mechanism for inter-process communication in UNIX [Roc85]. Fundamentally, a pipe is an I/O channel intended for use between two cooperating processes with a common ancestor. Named pipes, however, can be used for inter-process communication between two processes which do not have a common ancestor. One process writes into the pipe, while the other process reads from it. The system handles the synchronization between the processes and the buffering of data. If a process reads a pipe which is empty, it is blocked until data arrives; if a process writes into a pipe which is full, it is blocked until the pipe becomes partially empty.

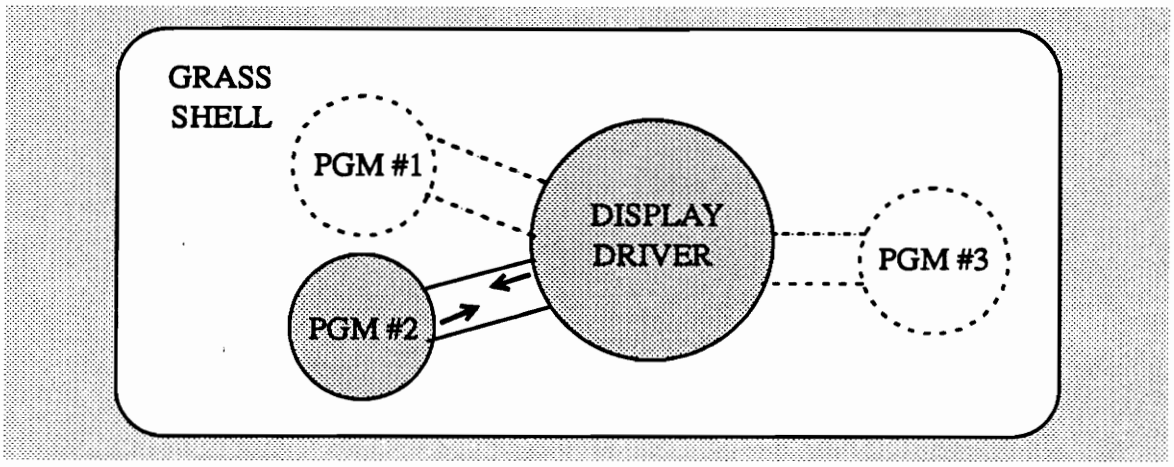

Figure 3-2. The GRASS process model. The program (PGM #2) shown shaded with the solid line is the only program that is currently executing. The display driver can interact with several programs (not simultaneously), through different pipes.

The graphics driver associates a pair of named pipes for use with each open dis-

play window. The programs that execute within the GRASS shell, retrieve data from the database and generate data as a result of the manipulation and analysis that they perform. This data, for it to be displayed, is transported over the pipe that the program utilizes to communicate with the graphics driver. The graphics driver reads the data from the pipe and uses the graphics primitives supported by the windowing system to display the data.

3.3.3 Capabilities of GRASS

GRASS is capable of handling geographic data obtained in different scales and resolutions. It uses the raster format for all of its image and geographical data processing. The raster data has attributes attached to each element in the data file. The vector format is used to store data which consists of distinct lines, such as those used by CAD packages. The system includes libraries of functions that provide efficient access to its database of raster and vector data. Besides the efficient data management capabilities, GRASS also has the capability for powerful map analysis and overlay. These include proximity analyses, weighted overlays, logical overlays, and neighborhood processing. The system can obtain input from the user interactively, and also display the results of the analyses using the array of graphics tools that it provides.

3.3.4 GIS Features Required in SISP

The site specific propagation prediction tool should incorporate the functionality common to most geographical information systems. These are listed below.

- Handle data at different scales and resolutions, obtained from different sources.
- Import spatial data, spatial objects, and non-spatial data or attribute data, available in standard formats.
- Organize and maintain the data in a structured database.
- Employ data compression techniques to reduce disk storage requirements.
- Display images of a user defined geographical region at any resolution.
- Export the data in popular, standard formats for use by the propagation prediction

programs and other software packages.

- Possess hardcopy-based image generation capability.

3.3.5 Modifications to GRASS programs

From a discussion of the GRASS process model in Section 3.3.2, it is evident that changes have to be made to these programs before they can be incorporated in other software packages. This section outlines the nature of changes that have been made to these programs, in order to incorporate the functional features of GISs, specified in Section 3.3.4, in SISP.

The GRASS process model is perhaps the most significant hurdle in incorporating GRASS programs, without modifications, in SISP. The user interface module of SISP invokes the functions which are used to display images in the graphics window. Hence programs that communicate with the display driver, using the inter-process communication mechanism of pipes, cannot fit directly within the framework of the software. Thus, such communication mechanisms have been removed in these programs.

As was discussed in Section 3.3.2, the functionality of GRASS is achieved by a set of programs executing within a common environment. Since each program is a separate entity that communicates with the display driver, the programs obtain state information, from the display driver, as part of the initialization procedure. This information is typically the number of graphics windows, the dimensions of the windows, the coordinates of the current geographical region, etc. When these programs are incorporated into a software package, there is no need to obtain state information explicitly, since this information is available in variables passed as parameters to the functions invoked. Thus the mechanism to obtain the state information has been removed to streamline the functions.

Another important issue is the memory management in the GRASS programs. Since each of these programs execute separately, there are many instances where the memory allocated in these programs is not explicitly freed. This is acceptable under the current GRASS structure, since the memory allocated by the programs is returned to the

system once the program exits. However if the programs are incorporated in a software package, the memory allocated in the routines has to be freed explicitly. Thus, several programs have been modified to emphasize good memory management techniques.

Yet another issue that needs to be considered is the overhead associated with programming using the interface library. The interface library that GRASS provides enables programmers to write programs that are consistent in structure. Since these routines are intended to be general purpose routines, there is a tangible overhead associated with using them directly in the software package. Thus many of these routines have been modified to eliminate redundancy and improve the efficiency.

The SISP raster toolkit has been developed using the modified GRASS programs as the backbone. New functions have been developed where they were either missing or inadequate. The sections that follow provide an overview and describe the features of the SISP raster toolkit.

3.4 SISP raster toolkit - An Introduction

The major motivation for the development of the SISP raster toolkit is to provide a set of functions that manipulate and display images. The toolkit has been developed using the Xlib procedural interface to the X Windows system [Jon89], [Nye88], [Nye88a]. The raster toolkit, consists of a library of C routines, which are used to compose and display images. A set of routines which can import and export raster data also forms a part of this toolkit.

The next few sections discuss some important attributes of the raster toolkit which impact its functionality: the format of the data, the coordinate system used, and the color model employed. Section 3.5 discusses the raster data format, the UTM coordinate system for referencing locations on the displayed maps is described in Section 3.6, and the RGB color model used to display images is discussed in Section 3.7. The rest of the chapter is devoted to a discussion of the capabilities of the toolkit.

3.5 SISP Raster Data Format

The format in which raster data is stored in the database determines the functionality of the routines that display the data. This section provides a description of the raster data file used in SISP (also referred to as the grid cell data file), which is the same as the one employed by GRASS [Wes91].

The grid cell data file is located in the database directory *cell*. In addition to the cell data file, there are a number of support files with the same name, located in different directories in the database. The database is structured according to the nature of information stored in the support files. Table 3-1 provides a listing of the directories with brief descriptions. A detailed discussion of the support file formats is provided towards the end of this section.

Table 3-1. Database directories for raster data files

Directory	Function
cell	Raster data file
cellhd	Header file
colr	Color palette files
cell_misc	Range file

3.5.1 Cell Data File

The cell data file can conceptually be viewed as a two-dimensional matrix of integer values. Each cell is stored in the file as one to four bytes of data. An N*x*M cell file will contain N rows, each row containing M columns or bytes of data. The physical structure of the cell file can take one of 2 formats: compressed or uncompressed.

- **Uncompressed format**

The uncompressed cell file looks like an N*x*M matrix. Each byte (or set of bytes for multibyte data) represents a cell of the map layer. The physical size of the file, in bytes, will be *rows*cols*bytes-per-cell*. Typical sizes of uncompressed terrain elevation files are

in the order of a couple of megabytes. The format of an uncompressed data file, containing 4 rows and 8 columns of data, is shown in Figure 3-3.

UNCOMPRESSED RASTER DATA FILE
86 86 87 86 86 78 80 86
88 78 86 84 84 84 86 90
90 96 96 96 96 96 94 88
76 76 78 78 78 80 82 84

Figure 3-3. Format of an uncompressed raster data file. 4 rows and 8 columns of data are shown in the figure.

- **Compressed format**

The compressed format uses a run-length encoding scheme to store the data [Hel87], [Rut72]. This technique helps reduce disk storage requirements and also helps in improving the performance of programs that read the data file by reducing the number of disk accesses. The degree of data reduction obtained as a result of the compression process is known as the compression ratio. This is a ratio between the quantity of compressed data and the quantity of original data. It has been shown that a good compression ratio can be achieved using run-length encoding schemes [Hel87]. Typical sizes of run length encoded terrain elevation files, covering an area of about one hundred square kilometers, are in the order of a few hundred kilobytes. The run-length encoding scheme is shown in Figure 3-4.

In this technique, sequences of the same data value are stored as a single byte repeat count followed by the data value as shown in Figure 3-4. If the data is single byte data, then each pair representing a compressed data value is 2 bytes. If the data is two byte

data, then each pair is 3 bytes. The rows are encoded independently; the number of bytes per cell is constant within a row, but may vary from row to row. Also if run-length encoding results in a larger row, then the row is stored without encoding. Since each row in the file may have a different length, there is an index to each row stored at the beginning of the data file [Wes91].

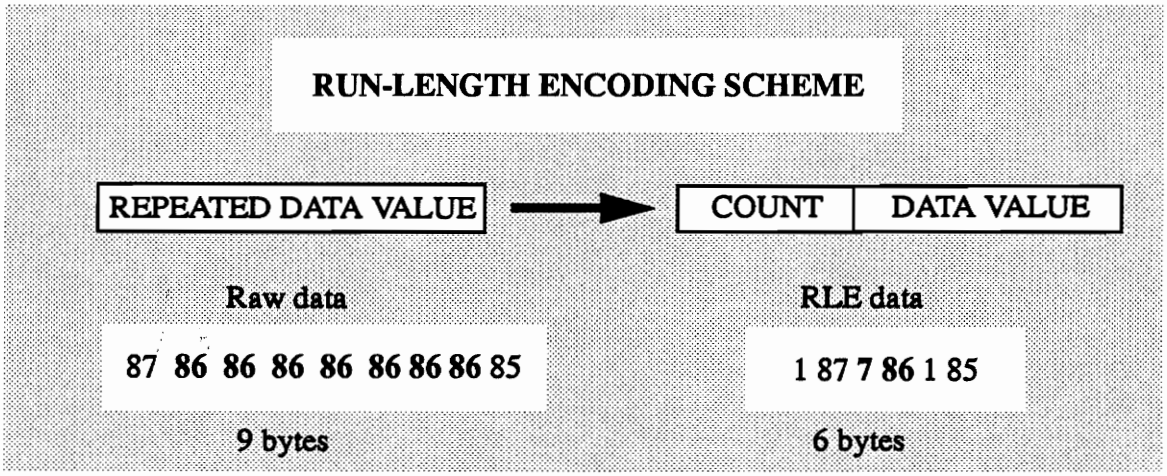

Figure 3-4. Illustration of the run-length encoding scheme. The elements that are involved in the encoding have been highlighted.

Figure 3-5 shows the uncompressed data file and the corresponding compressed data file.

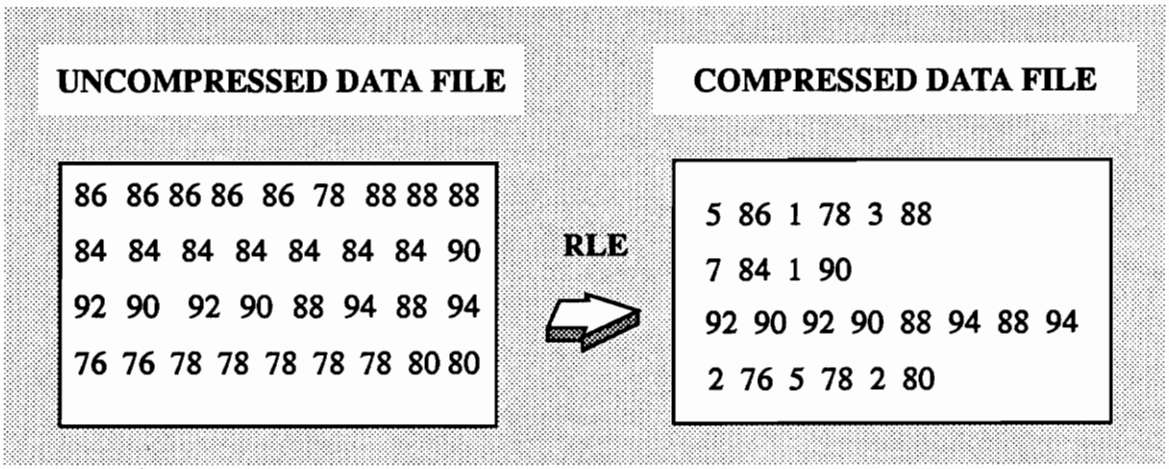

Figure 3-5. Format of a compressed data file. The uncompressed data file is also shown. The third row has not been encoded since encoding increases the number of bytes.

It is apparent from the discussion of the format of the cell data file, that the file contains the information needed for analysis. However the cell file itself does not contain information about the geographical region for which the data is relevant, the resolution of the data set, the range of data values present in the file. The support files provide this information. A discussion of the header, color and range files follows.

3.5.2 Header file

The ASCII header file located in the directory *cellhd* contains locational information for the data set, and the resolution of the data. The format of the header file is shown in Figure 3-6.

HEADER FILE

proj:	1
zone:	17
north:	4122585.5
south:	4108665.5
east:	555520.5
west:	544360.5
e-w resol:	30.0
n-s resol:	30.0
format:	0
compressed:	1

Figure 3-6. Format of the cell header file.

The projection field *proj*, specifies the type of cartographic projection. The possible values it can have are: 0 to indicate unreferenced locations and 1 to indicate UTM. The field *zone*, specifies the UTM zone. The *north*, *south*, *east* and *west* fields specify the coordinates of the geographical boundaries of the cell file. These lines do not pass through the center of the cells at the edge of the map, but along the edge of the map. The grid resolution fields, *n-s resol* and the *e-w resol* describe the size of each grid cell in the map in physical measurement units (e.g., meters in a UTM database). The *n-s resol* is the length of a grid cell along the north-south direction and the *e-s resol* is the distance along the east-west direction. The *format* field describes the number of bytes per cell (data point) that are required to represent the grid cell data. A zero value for this field indicate 1 byte per cell. The last field in the header file *compressed*, contains a 1 or a 0 to indicate if the

file is compressed or not.

3.5.3 Color file

The *colr* directory contains ASCII files that store data for user defined color palettes. The color table contains one line of a color description for each category of data, including the "no data" category. The colors are represented as levels of red, green, and blue, where 0 represents the lowest intensity and 255 represents the highest intensity. The format of the color file is shown in Figure 3-7. The numbers which follows the '%' character in the first line of the file, indicate the first data value and the last data value which have a color. The range of data values is split into intervals. The remaining lines are the RGB values for the limits of these intervals. The values in the file separated by colons, represent the data value, and the red, green and blue levels respectively, which are associated with it.

COLOR TABLE FILE

```
% 1   423
1:255:255:0        85:0:255:0
85:0:255:0         170:0:255:255
170:0:255:255      254:0:0:255
254:0:0:255        339:255:0:255
339:255:0:255      423:255:0:0
```

Figure 3-7. Format of the color file.

An alternative format for the color file exists and is shown in Figure 3-8. The numbers which follow the '%' character in the first line of the file, indicate the first data value

and last data value that have a color. Since this range of data values is less than 255 (the number of color cells in the colormap), each data value in this range is assigned a specific RGB value. The alternate file format is indicated by the "%%" character string in the second line of the file. Thus the number of entries in the file equal the range of data values. The values in the file, delimited by colons, represent the offset from the lowest data value, and the red, green and blue levels respectively, associated with each data value in the range.

COLOR TABLE FILE

```
% 82 88
%%
1:255:255:0
2:0:255:255
3:0:255:255
4:255:0:255
5:255:1:235
6:255:0:0
```

Figure 3-8. Alternate format of the color file.

The color file provides the user the flexibility to associate a color scheme, other than the default, with the data set. This color scheme is used to encode the data, when the data is displayed.

3.5.4 Range file

The range file contains information about the maximum and minimum data values

in the cell file. This file is located in the directory *cell_misc* as *cell_misc/name* where *name* is the name of the corresponding cell file. This ASCII file contains one line with four integer values. These represent the maximum and minimum negative values, and the maximum and minimum positive values. If the cell file contains only positive values then the first pair of numbers will be zero. If the cell file contains no positive values then the second pair of numbers will be zero.

From the preceding discussion of the raster data format, it is apparent that the data is organized such that the different pieces of information are stored as distinct entities in the database. This database structure ensures that routines only access information relevant to the functionality that they provide. For instance separating the color file from the cell data file, ensures that the routines that setup the color translation table, used for the display of images, do not access the cell data values.

3.6 The UTM Coordinate System

In order to locate points relative to one another, it is necessary to employ the concepts of direction and distance. A universal or general coordinate system has to be established in relation to some unique reference or starting point. If such a point is designated, then the location of every other point can be stated in terms of a defined direction and distance from it.

In recent years, the use of plane, rectangular coordinate systems has become widespread. For precise referencing and directional relationships, it is necessary to develop a plane coordinate system using a map projection with the desired characteristics. A plane coordinate system which has been adopted for referencing of satellite imagery, and other applications requiring precise positioning is the Universal Transverse Mercator (UTM) system [Col80], [Cam84], [Mal92]. SISP uses the UTM coordinate system for referencing geographical data and locating various map features. This system is also used in topographic maps obtained from the USGS (United States Geological Service).

The UTM system uses the Gauss-Kruger version of the transverse Mercator and

provides coverage between 80° S and 84° N. The UTM system involves establishing 60 north-south zones, each 6° of longitude wide. Each zone is numbered, beginning with zone 1 at the 180th meridian near the International Date Line, with zone numbers increasing to the east. Most of the United States is included in the zones 10 through 19 as shown in Figure 3-9.

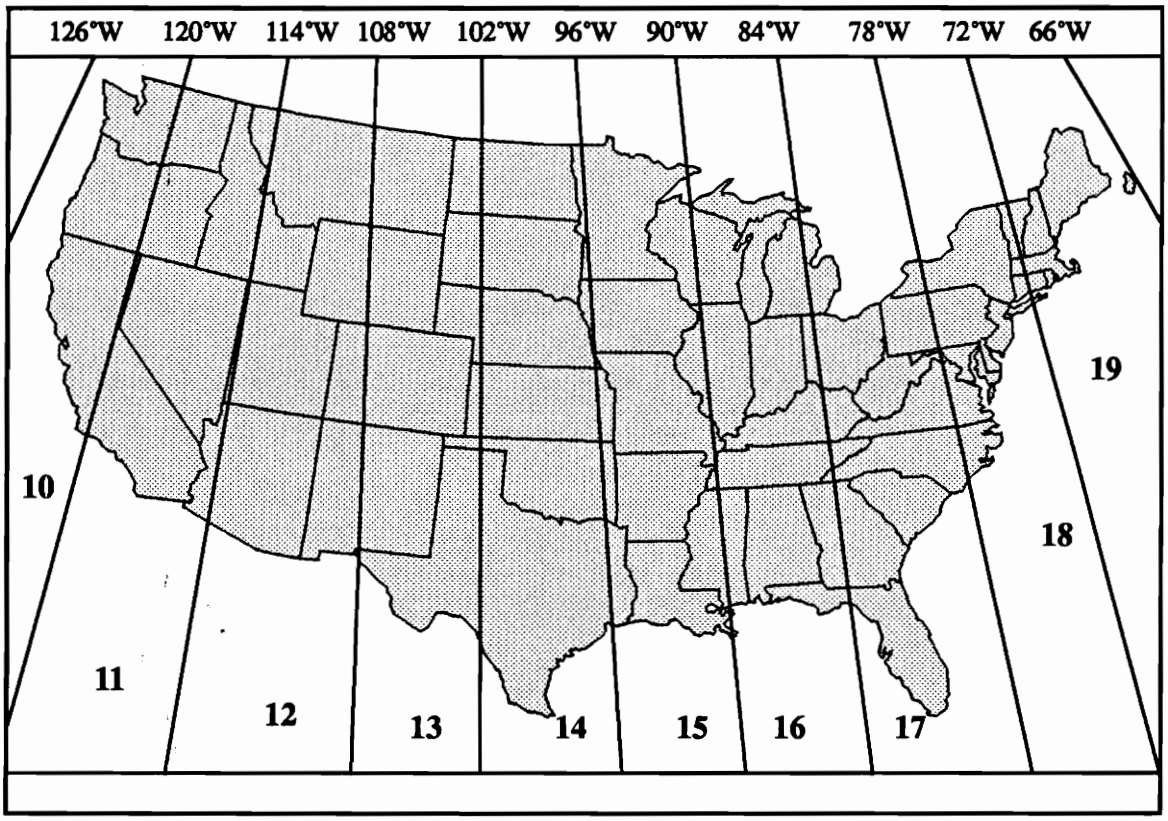

Figure 3-9. UTM Zones in the contiguous United States [Col80].

In this system, each zone is flattened onto a plane surface and a square grid is superimposed upon it. The two most important features of the zones are the equator, which runs east-west through its center, and the central meridian, a north-south line through its center. False origins (so called since they lie outside the UTM zone) are established 500, 000 m west of the central meridian of each zone as shown in Figure 3-10. Since the origins for each zone lie outside and to the left of the zones, the coordinates for any location

within the zone is positive. In the northern hemisphere these origins are on the equator, but in the southern hemisphere they are 10, 000, 000 m south of the equator. Within any zone, the distance in meters from the origin, in the north-south direction, gives the northing of any point. Similarly the distance in meters from the origin, in the east-west direction, gives the easting of that point. Thus any point in the zone may be referred to by its zone-number, its easting, and its northing. A complete citation for a location in zone 16 in the northern hemisphere with easting 423, 870 m and northing 4, 717, 510 m would be given as: UTM Northern Hemisphere, zone - 16, 423 870, 4 717 510.

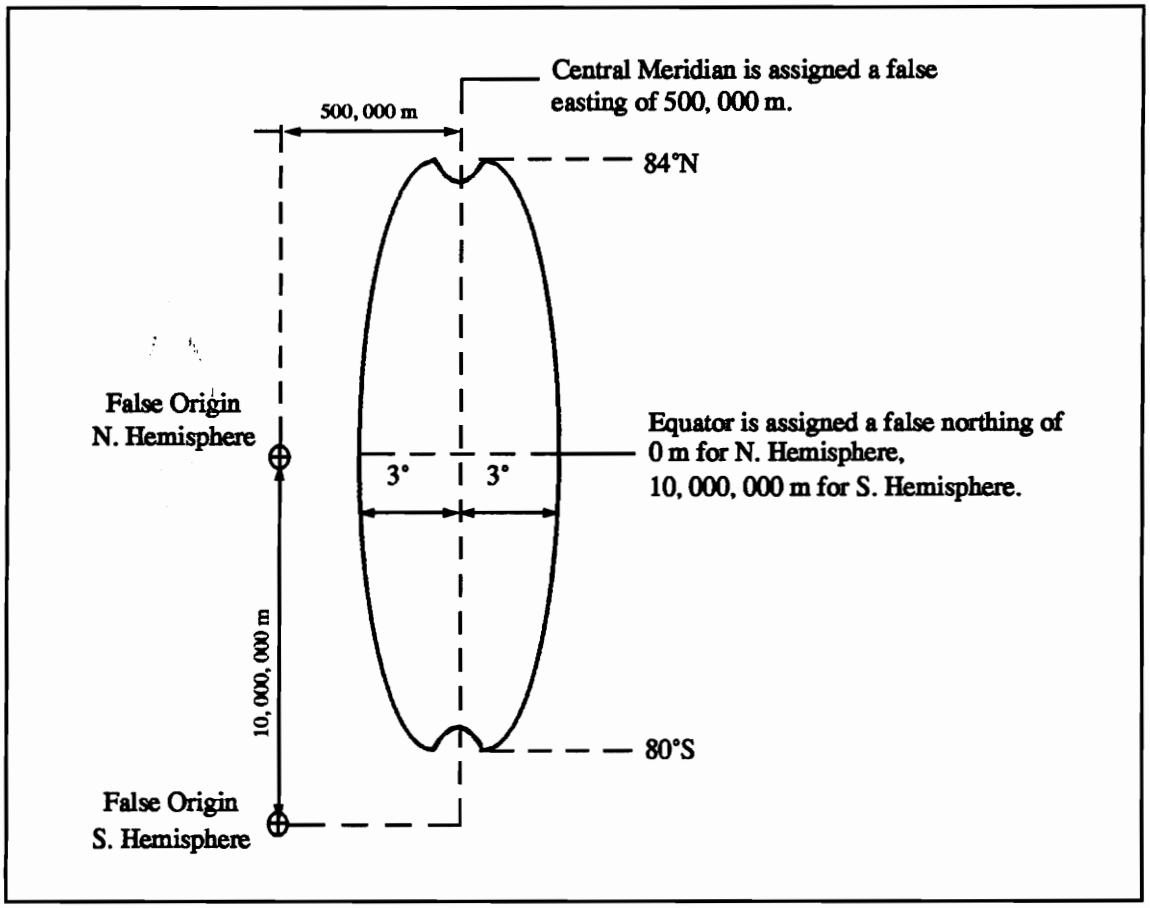

Figure 3-10. A typical UTM zone. A false origin is established 500, 000 m of the central meridian. In the northern hemisphere this origin is located on the equator. The zone is not drawn to scale. The shape is exaggerated here for clarity.

The greatest advantage of the UTM system over the geographical coordinate system (latitude/longitude) is its reliability. This is primarily because in the UTM system, the coordinates are stated in linear, decimal units, rather than in angular, non-decimal ones. The system also has other benefits, such as simplicity of notation and greater precision, when compared to the geographical coordinate system.

3.7 The RGB color model

In order to design an effective presentation of the raster data, the data values are encoded with different colors, which are then displayed on the screen. The color model that is chosen to display the image determines the functionality of the routines that are used to perform the operation. SISP utilizes the RGB (red/green/blue) color model. A brief discussion of the color model used is necessary for a clear understanding of the display operation, and this has been presented in the paragraphs that follow.

The RGB color model describes an additive color system and is typically used in display terminals. Subtractive color models such as CMY (cyan/magenta/yellow) are used for printing on paper. In an additive color system, color is generated by mixing rays of light of different frequencies. In practical terms, additive mixing involves three primary colors. The usual additive primaries are red, green and blue. Since there are three colors, they can be assigned to the three perpendicular axes necessary to represent points in three dimensional space. Thus there is a red axis, a green axis, and a blue axis and any color can be represented in this 3-dimensional space as a tuple of a red, green, and blue values.

Mathematically, the model is represented by denoting the absence of a color with a brightness of 0, and denoting the greatest possible brightness as 1. Thus the model can be represented by the unit cube in Figure 3-11 with the values at the vertices denoting the brightness of the red, green, and blue colors at that point [Fol90], [Gla90].The corner of the cube where the color axes start, is the origin, and has the coordinates [0, 0, 0]. This would be perceived as black. The opposite corner, one unit away on each axis represents the color in which red, green and blue are at the brightest. This is perceived as white.

The additive primaries are shown in Figure 3-11, represented by the vertices labeled R, G, and B. A combination of blue and red is perceived as magenta, a combination of blue and green is perceived as cyan, and a combination of red and green is seen as yellow. The secondary colors cyan and magenta are also shown in Figure 3-11, represented by the vertices labeled C and M. Thus, in the RGB model every possible color can be represented by a set of RGB values.

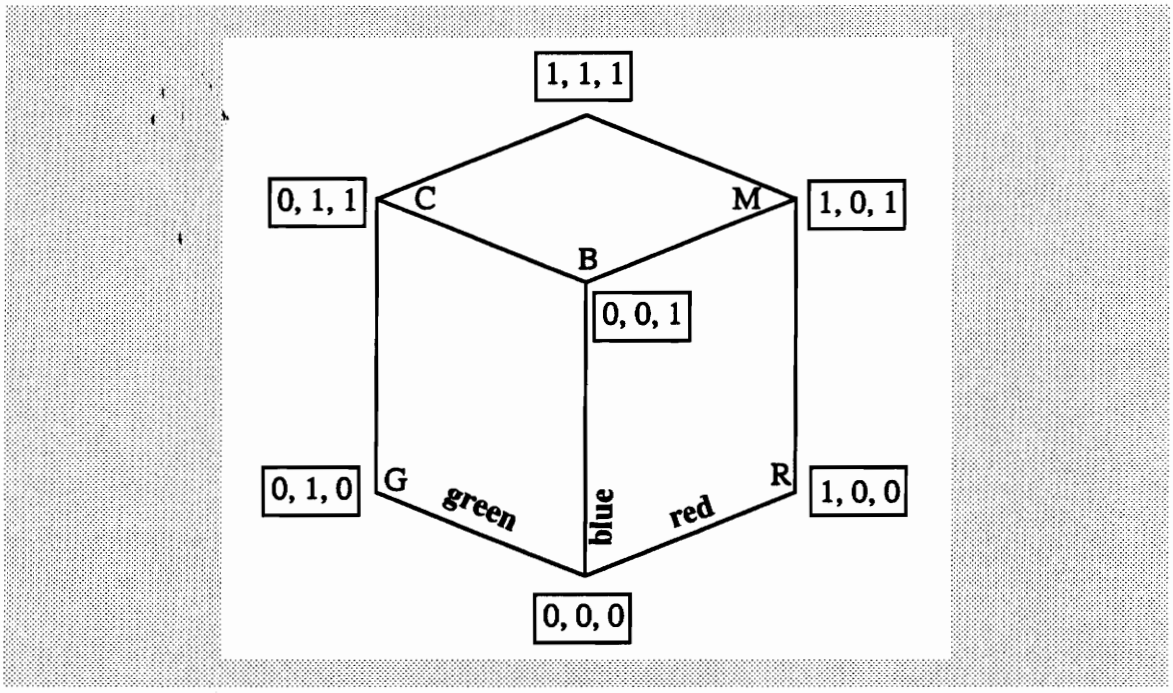

Figure 3-11. RGB color cube. The RGB color model is represented as unit cube, with the red, green and blue color axes, with 1 denoting the maximum brightness and 0 denoting the minimum brightness.

3.8 Initialize display window

The graphics window is the principal user interaction object in SISP. The attributes of the display window have to be set before the display routines can redirect the output to it. The objectives of the initialization procedure are:

- To initialize system variables used by all display routines
- To set the attributes of the display window to indicate the colormap that needs to be used to interpret the pixel values, to ensure the contents of the window are saved when the window is obscured by other windows or iconified, and to ensure speed of redraw operations.
- To allocate a colormap for the display operations
- To allocate pixmaps for drawing operations

The initialization procedure is invoked when a window is created. The attributes of a window can be specified at the time it is created or can be modified after the window has been mapped. Mapping a window marks the window eligible for display by the window manager. When a window is created, the attributes of the window are set to default values. The initialization procedure modifies default values of attributes that are needed to achieve the objectives outlined above. The remainder of this section discusses the functionality of the initialization procedure.

The user interface module passes the widget identifier of the display window to the initialization procedure. The Xlib functions and macros, which are called within the initialization procedure, require information about the server connection, and resource identifiers as parameters [Nye88a]. This information is extracted from the widget identifier using the Xt library. The X Toolkit Intrinsics, Xt, is a library layered on Xlib, which provides the functionality from which the widget sets are built [Ase90], [Ore88], [You90]. The function *XtDisplay()* is used to initialize, *Display*, a pointer to the structure that stores information about the server connection. The function *XtWindow()* initializes, *Window*, a resource identifier of the display window. Other parameters required by the Xlib functions are initialized by calling the appropriate Xt functions and macros. The attributes of the window are now modified using appropriate Xlib functions.

The contents of the display window need to be saved when the window is obscured, iconified, or unmapped. A feature of some X servers called backing store auto-

matically maintains the contents of the window while it is obscured or unmapped. The *backing_store, backing_planes,* and *backing_pixel* are attributes of the window which control the backing store feature. The *backing_store* attribute determines when, and if a window's contents are preserved by the server when the window is obscured or unmapped. The *backing_store* attribute can have the following possible values: *NotUseful, WhenMapped, Always. NotUseful* advises the server that saving the contents of the window is unnecessary. *WhenMapped* advises the server that the contents of the obscured regions of the window need to be maintained when the window is mapped. *Always* advises the server that the contents of the window need to maintained even when the window is unmapped. The *backing_planes* attribute specifies a mask that indicates which planes of the window need to be preserved in the backing store. The *backing_pixel* attribute specifies a pixel value to be used in planes that are not backed up.

The mechanism which is used to display images in the display window needs to be presented before a description of the alternatives considered for setting the *backing_store* attribute of the window can be presented. When a user desires to display a new image in the display window, the window is not destroyed. The window is unmapped and is mapped again when the user chooses the new image to be displayed. The *ExposeEvent,* generated when a window is mapped, is used to invoke the function that displays the image. Thus the display window is not created and destroyed each time a new image needs to be displayed.

The Xlib function *XChangeWindowAttributes()* with the appropriate bitmask is used to set the *backing_store* attribute for the window. When the *backing store* for the window is set to *Always,* the contents of the window are retained even when the window is unmapped. As explained earlier, when the user desires to display a new image the window is unmapped. However, when the window is mapped for displaying the new image, the function that displays the image is not invoked. The reason for this being that the server retains the contents of the window and restores these contents when the window is mapped again. The ExposeEvent that is normally generated is masked, and hence the rou-

tine to display the image is never invoked. To overcome this problem the *backing_store* for the window is set to *WhenMapped*. With this option, the server maintains the contents of regions of the display window that are obscured by other overlapping windows. This information is restored when the obscured regions of the window become visible. If the display window is unmapped or if it is iconified, the server does not store the contents of the window. However, an *ExposeEvent* is sent to the application, when the window is mapped again, which is used to invoke the function to display images.

The initialization procedure creates pixmaps, that are used as a backup, to store the contents of the display window during various drawing operations. A pixmap is an array of pixels and are maintained in off-screen memory. The contents of the pixmaps do not appear on screen unless they are copied to an existing visible window. When a display operation is performed, the routines draw to the visible display window and the currently active pixmap. If the same image needs to be redrawn the function to display the image is not invoked, instead the image is copied from the pixmap to the display window. Thus the pixmaps are used to provide a fast redraw operation.

The initialization procedure also creates a colormap, used to lookup pixel values of the image to be displayed in the display window. Since the display of raster images requires the use of several colors, color cells are allocated from the default RGB colormap of the server. In some situations, it might not be possible to allocate all the desired colors from the default colormap. In such situations a new colormap, called a virtual colormap is created. This colormap is specified as the *colormap* attribute, for the top-level window of the software, using the Xlib function *XChangeWindowAttributes()*. Specifying a colormap as the default colormap, ensures that this colormap is installed by the server when, the pointer focus is in the display window. This leads to color flashing when the focus moves in and out of the application. Additionally the displayed images appear in false colors when the pointer focus is outside the display window. The alternative to this approach is to use the Xlib function *XInstallColormap()* to explicitly install the colormap for the window specified. Calling this function during the initialization process, does not serve the desired

purpose since the server installs the default colormap whenever the pointer focus moves to a window outside the display window. Thus the application needs to keep track of *FocusIn* events, and be responsible for re-installing the colormap when the pointer focus moves back to within the display window.

The initialization procedure initializes variables such as the height, width, etc. of the display window which are used by all the drawing operations.

3.9 Displaying Raster Data

The routines described in this section provide the capability to display raster images and are derived from GRASS source code. A resume of the various components of this operation, in their functional order, is given below.

- Verify the existence of the necessary database elements.
- Check for the existence of a user defined palette or color file.
- Read the range file if the color file does not exist.
- Form class intervals using the data and assign RGB values to the limiting values of each class interval.
- Read the header file and compute the coordinate transformation factors.
- Read the raster data file and decompress it, if stored in run-length encoded format.
- Assign RGB color values to the data values, using the class intervals and interpolation techniques.
- Perform conversions from the data coordinates to the display coordinates.
- Determine, from the colormap allocated, the exact pixel values for each RGB tuple, and display the image

As described in Section 3.5, the raster data is stored in the SISP database as a set of data elements in different directories. Accordingly, the first step in displaying the raster

data, is to check for the existence of the support files for the data file specified by the user. The routine checks if the header file and the range file exists. If these files do not exist, vital information necessary for the display of the data set is unavailable and the routine returns an error value.

After the routine verifies the existence of the support files, it checks for the presence of a color file. The color file, whose format has been discussed in Section 3.5, provides information about the data set and the corresponding RGB values. The color file is optional and may be present in the *colr* directory of the database.

If the color file exists, the routine reads the color file and interprets the information contained within, based on the format of the file. This information is stored in a structure *Colors*. The template of the structure and its member elements is shown in Figure 3-12. The color information related to each data set is stored in the *_Color_Info_* structures *fixed* and *modular*. The format of the color file determines the structure that is used to store the information. For a color file of the format shown in Figure 3-7, the structure *modular* is used to store the information, while for a color file of the format shown in Figure 3-8, the structure *fixed* is used to store the information. Section 3.5 contains an exhaustive description of the color file formats.

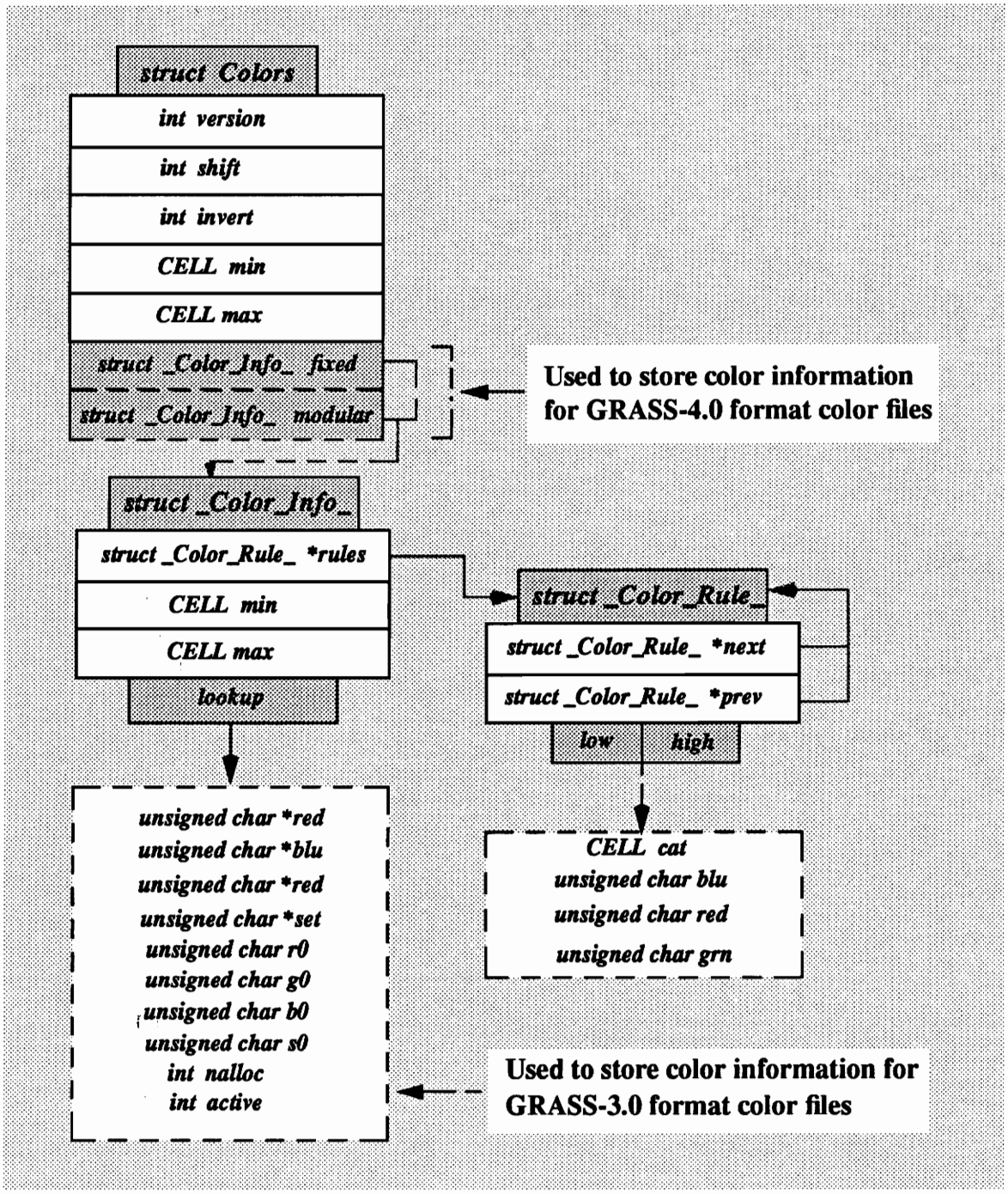

Figure 3-12. Template of the *Colors* structure. The shaded blocks are member elements which are structures themselves. The solid arrows indicate structure pointers.

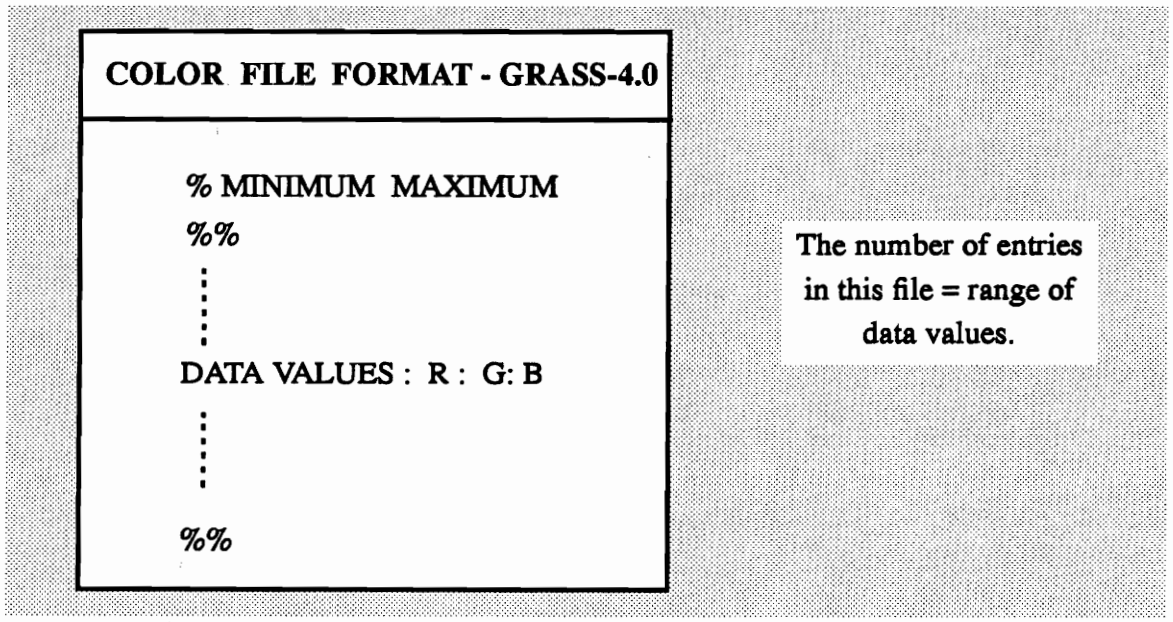

Figure 3-13. Format of a GRASS-4.0 color file. When the color file in the *colr* directory is of this format, the information is stored in the structure *modular*. Section 3.5 provides a comprehensive discussion of color file formats.

If the color file does not exist in the database, the routine devises a color encoding scheme which is used to compose the image. The routine uses the color quantization table given in Table 3-2, which is used to quantize the data values present in the data file. The procedure of color quantization consists of reading the range file and splitting the range of data values into class intervals. The RGB values in the quantization table are then assigned to the limiting values of these class intervals. There are several different approaches to the choice of class intervals [Kea89]. The equidistant method, in which the range of data values is split into equidistant intervals, has been chosen. Six class intervals are formed and the RGB values in the color quantization table are assigned to the limiting values of each class interval

Table 3-2. Color quantization table with 6 levels

R	G	B	Color
255	255	0	Yellow
0	255	0	Green
0	255	255	Cyan
0	0	255	Blue
255	0	255	Magenta
255	0	0	Red

The class intervals and the corresponding RGB values are stored in a doubly linked list. The template of the structure *_Color_Rule_* which is the element of the list is shown in Figure 3-14. The elements in the doubly linked list are inserted by the routine *AddColorRule()* and is used to color code the data values.

Following the completion of initialization procedures for color quantization, the routine reads the header file for the data set. As discussed in Section 3.5, the header file contains information about the geographical coordinates of the region and the resolution of the data. This information is utilized to compute coordinate transformation factors, to convert between geographical coordinates and screen coordinates, and between data array indices and screen coordinates.

The transformation factor to translate between the geographical coordinates and the display coordinates is computed such that the ratio of the scaling factors along the vertical and horizontal axis is unity. This ensures that the displayed image is isotropic. This is achieved by computing the transformation factors for the vertical and horizontal axes and choosing the smaller of the two values as the transformation factor. Since the smaller of the transformation factors is applied along both the axes, the displayed image is smaller than the dimension of the window along one axis. Hence the image has to be centered in the graphics window at the time it is displayed.

Subsequent to computing the coordinate transformation factors, the data values are read from the raster data file one line at a time, decompressed, and stored in main memory. This allows the data values to be accessed with minimum delay, since the data need not be read off the disk each time. This is of vital importance for the continuous attribute tracking feature that is provided by the toolkit. The routines that read the data file and uncompress the data are functions from the GRASS function library, that is a part of the standard distribution. This ensures that changes to the data format can be handled easily by replacing these library functions with functions provided with the new distribution.

The assignment of color values to the data values is handled in two different ways. If the range of values in the data set is less than the number of entries in the colormap (which is typically 256), the color values are assigned to all the data values in the range at the time of setting up the color encoding scheme. If the range of values exceeds the number of entries in the colormap, the assignment of color values is performed at the time of displaying each row of raster data. The color encoding is performed by the routine *G_color_lookup()*. The routine searches the linked list for each data value to determine the class interval it lies in. Once the interval has been determined the actual RGB values are computed by the technique of linear interpolation between the RGB values assigned to the upper and lower limits of the class interval.

Once the RGB values have been assigned the routine *LookupPixelValue()* is used to determine the appropriate pixel value to display. This is performed by using the RGB values as an index into the colormap allocated. The pixel values, for a row of raster data, is copied to an *XImage* structure and displayed in the window using the Xlib function *XPutImage ()*. The image is also drawn to the currently active pixmap. The image drawn to the pixmap is necessary for a fast redraw operation, for instance, while reverting to the original image after a zoom operation on the image. Thus if the application needs to redraw the displayed image, it copies the image from the pixmap to the window. The memory allocated for the *XImage* structure data is freed when the entire raster image has been displayed. Since the size of the *XImage* data changes with the zoom factor that is in effect,

the, memory, used to store the image data is freed after every display operation.

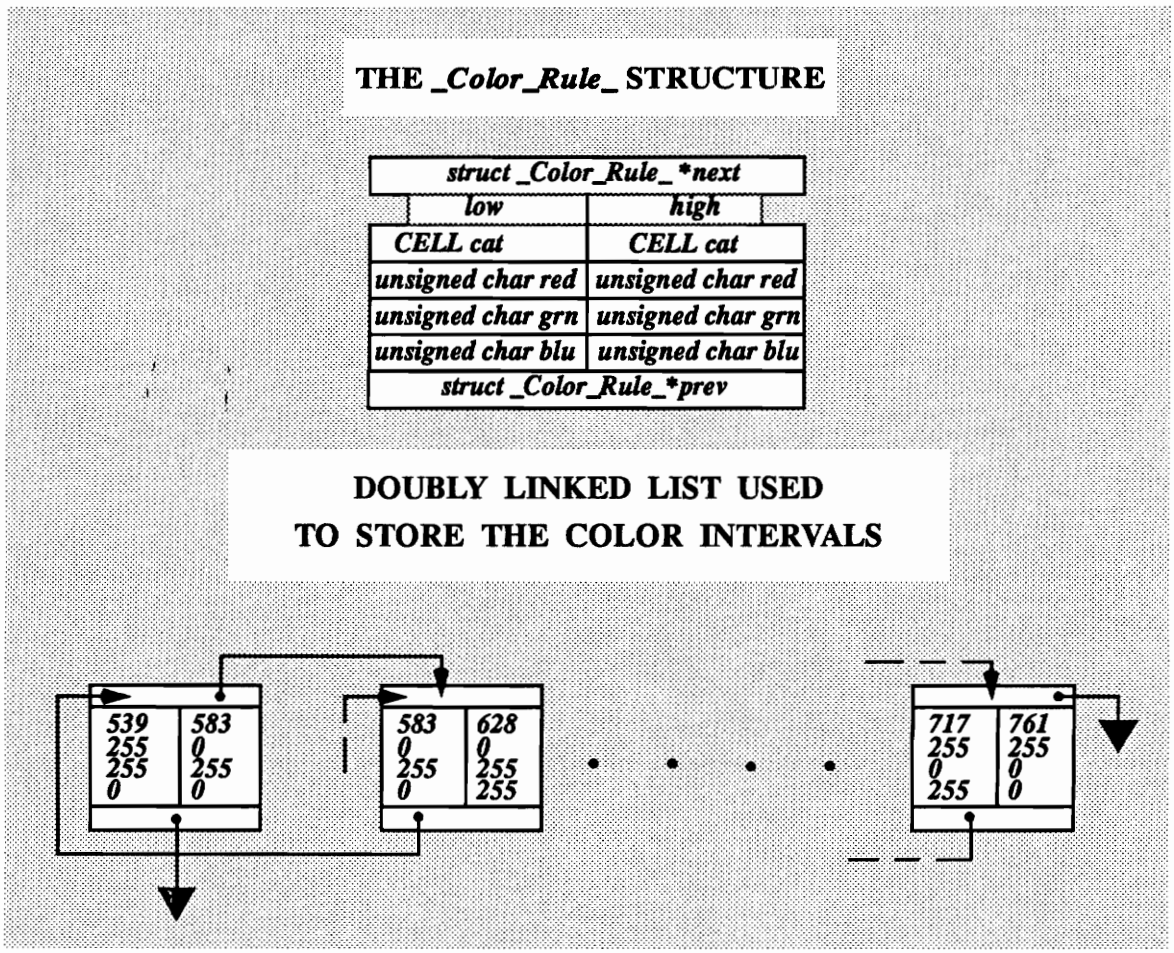

Figure 3-14. Data structure that is used to store the color encoding scheme. This is a doubly linked list, in which the elements contain the limits of the intervals of data values, and the corresponding RGB values. The figure shows one such list, for a data set whose range is between 539 and 761. These elements are inserted into the list by the routine *AddColorRule()*.

The various operations involved in displaying raster data is shown in Figure 3-15.

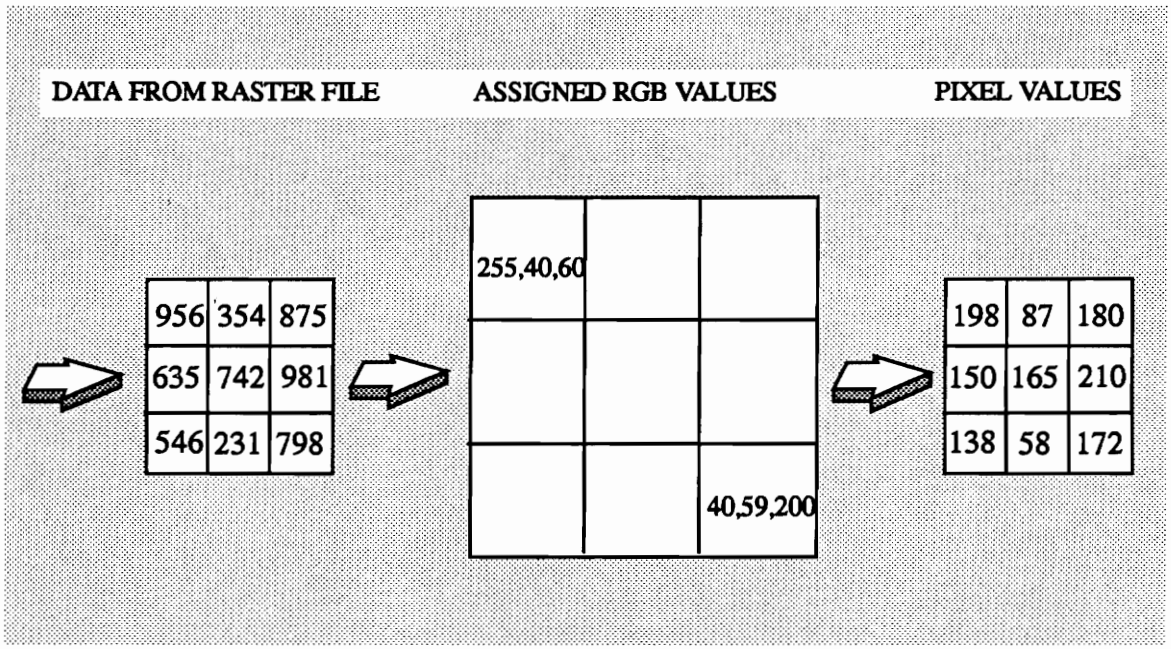

Figure 3-15. Color coding of raster data. Data is read from the raster file and uncompressed. Color values are assigned using the color encoding scheme devised. The RGB values are used as an index into the colormap to determine the actual pixel values that are displayed.

3.10 Displaying the legend

This function is used to annotate the image displayed, by providing a legend for the colors in the image. It provides an estimate of the data values that have been encoded and displayed. The range file for the raster data is read and the minimum and maximum values in the data set is determined. This range is then divided into ten equidistant intervals. The limits of these intervals are stored in an array. The data in this array is similar to a row of raster data read from the data file. The color encoding scheme that was stored in the linked list, while displaying the image, is used to assign RGB values to the data values in the array. The pixel values corresponding to these RGB values are obtained by performing a lookup to the colormap, that was allocated during the initialization procedure. Thus

an array containing the pixel values is obtained.

The Xlib drawing primitives are used to create the displayed legend. The function *XDrawRectangles ()* is used to create the required number of rows in the legend. In an iterative procedure, each pixel value in the array is set as the foreground color for the window, and the Xlib function *XFillRectangle()* is used to color one row of the legend. The Xlib function *XDrawString()* is used to display the data value which each color in the legend represents. The data values displayed in the legend are representative values from the range of values for the data set and may not actually exist in the data file. Thus the legend only provides an approximate indication of the color encoding scheme employed for the data set.

3.11 Conversions between coordinate systems

The advantages of using a planimetric coordinate system such as UTM, have been stated in Section 3.6. Very often, terrain elevation and feature data for a geographical region are obtained from independent sources. Although this data is geo-referenced, it could be referenced to entirely different coordinate reference systems. For instance, terrain elevation data obtained from the USGS could be referenced to the UTM coordinate system, while building data obtained from a different source could be referenced to the geographical coordinate system. In order to obtain a coherent view of the database, the data from these sources have to be referenced to a single coordinate system. Moreover, locations of transmitters and receivers, during measurement campaigns, are often determined using global positioning systems (GPS), in which locations are referenced to the geographical coordinate system. Thus the capability of transforming geo-referenced data between the various coordinate systems is essential.

As stated earlier, the native coordinate system for SISP is the UTM system. Since the geographical coordinate system is also widely used, routines have been developed to perform transformations from the UTM system to the geographical coordinate system and vice versa. These routines are based on Meade's nested solutions [Mea87] to Redfearn's

equations [Rea48]. Meade's solutions reduce the risk of overflow conditions, which could result from progressively raising terms in Redfearn's equations to higher powers. The conversions are performed using the Clarke 1866 spheroid (NAD 27) as the reference spheroid. A detailed discussion of the UTM system and the choice of the reference spheroid is provided in [Mal92]. The equations for converting from the geographical coordinate system (φ, λ) to the UTM system (N, E) are given below.

$$
\begin{aligned}
N &= k_0 \cdot m + A_2 L^2 \left[1 + L^2 \left(A_4 + A_6 L^2\right)\right] \\
E &= A_1 L \left[1 + L^2 \left(A_3 + L^2 \{A_5 + A_7 L^2\}\right)\right] \\
L &= \lambda \cdot \cos\varphi \\
A_1 &= k_0 \cdot \nu \\
A_2 &= \frac{1}{2} k_0 \cdot \nu \cdot t \\
A_3 &= \left(1 - t^2 + \eta^2\right)/6 \\
A_4 &= \left[5 - t^2 + \eta^2 \left(9 + 4\eta^2\right)\right]/12 \\
A_5 &= \left[5 - t^2 \left(18 - t^2\right) + \eta^2 \left(14 - 58t^2\right)\right]/120 \\
A_6 &= \left[61 - t^2 \left(58 - t^2\right) + \eta^2 \left(270 - 330t^2\right)\right]/360 \\
A_7 &= \left[61 - t^2 \left(479 - 179t^2 + t^4\right)\right]/5040
\end{aligned}
$$

where φ, λ are the latitude and longitude, N and E are the northing and the easting, k_0 is the scale factor along the central meridian, m is the meridional arc length of the reference spheroid, ν is the transverse radius of curvature of an ellipsoid, t is the tangent of the latitude, and η is the ordinate of the radius of the curvature.

Similarly, conversion from the UTM system (N, E) to the geographical coordinate system (φ, λ) is accomplished by the set of equations that follow.

$$\varphi = \varphi' + B_2 Q^2 \left[1 + Q^2\left(B_4 + B_6 Q^2\right)\right]$$

$$L = Q\left[1 + Q^2\left(B_3 + Q^2\{B_5 + B_7 Q^2\}\right)\right]$$

$$\varphi' = \text{atan}\,[a\tan u/b]$$

$$Q = (E/k_0)\cdot \nu'$$

$$B_2 = -t'\left(1 + \eta'^2\right)/2$$

$$B_3 = -\left(1 + 2t'^2 + \eta'^2\right)/6$$

$$B_4 = -\left[5 + 3t'^2 + \eta'^2\left(1 - 9t'^2 - 4\eta'^2\right)\right]/12$$

$$B_5 = \left[5 + t'^2\left(28 + 24t'^2\right) + \eta'^2\left(6 + 8t'^2\right)\right]/120$$

$$B_6 = \left[61 + t'^2\left(90 + 45t'^2\right) + \eta'^2\left(46 - 252t'^2 - 90t'^4\right)\right]/360$$

$$B_7 = -\left[61 + t'^2\left(662 + 1320t'^2 + 720t'^4\right)\right]/5040$$

where φ' is the foot-point latitude and u is the reduced latitude.

With data referenced to the UTM coordinate system, problems will be encountered when a geographical area straddles two adjacent UTM zones. From a discussion of the UTM system, provided in Section 3.6, it is evident that the UTM coordinates repeat in every zone. Distance computations within this region would be in error unless this feature of the coordinate system is accounted for. A possible solution to this problem is shown in Figure 3-16.

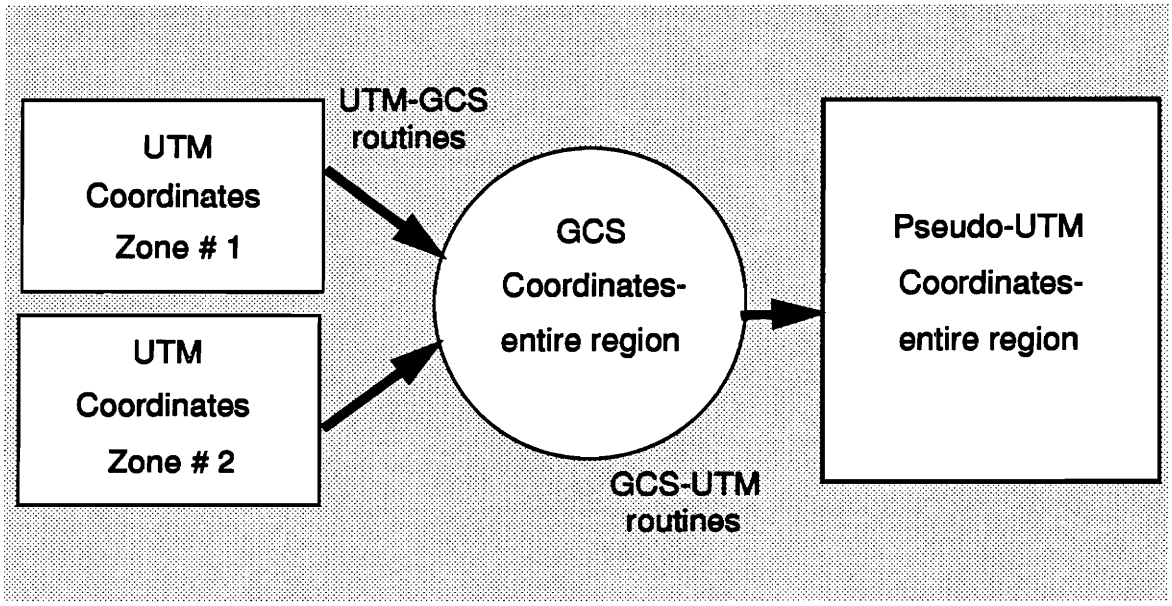

Figure 3-16. Solution to the UTM zone straddling problem.

The central idea of this solution is to establish a pseudo-UTM coordinate system, in which the entire geographical area falls within a single zone. First, the geographical coordinates for the entire area are obtained using the UTM to geographical coordinates transformation routines that have been developed. The central meridian for the pseudo-zone is established at the mid-longitude of the geographical region. The coordinates referenced to the pseudo-UTM coordinate system are then computed utilizing the geographical coordinates to UTM routines that have been developed. These coordinates are utilized for estimating distances within this area. It must be emphasized, that this pseudo-UTM coordinate system is established only for the purpose computing distances within this area, and does not affect the coordinates displayed to the user.

3.12 Continuous coordinate tracking

This function provides the geographical coordinate information, corresponding to the position of the mouse pointer, in the displayed image. The position of the mouse is continuously tracked and screen coordinates, indicating the position of the mouse pointer

within the display window, are obtained. The coordinate transformation factors, computed prior to displaying the image, are applied to convert the screen coordinates into Universal Transverse Mercator Projection (UTM) coordinates. SISP uses the UTM system as the basic coordinate system, since the terrain data supplied by the USGS references locations based on this system. The UTM system is explained in detail in Section 3.6 of this chapter.

The coordinates can also be displayed in the geographical coordinate system (GCS) format, as latitude-longitude. Since SISP primarily uses the UTM coordinate system.,the conversion between UTM and GCS is performed only for display purposes.

3.13 Continuous attribute display

This function provides the exact data values which were encoded to create the image. An estimate of the coloring scheme employed for a data set can be obtained from the legend displayed for the image. However an approximate data value might not be appropriate for every application. For instance, the height of a transmitting or receiving antenna above sea level is an important parameter in propagation prediction. Thus, when locating a transmitter or receiver, the exact elevation of the underlying terrain at any point in the displayed image needs to be provided.

In order to provide the capability of continuous attribute tracking, the position of the mouse is tracked and its position in screen coordinates is obtained. The coordinate transformation factors are applied to convert the screen coordinates to indices into the data array. The data value stored at that location in the array corresponds to the data value at the location of the mouse pointer, on the displayed image. Storing the data values in main memory improves the performance of the function, since it eliminates the need for expensive disk accesses are need to retrieve the data. The function returns a value of zero if the mouse pointer is outside the region in which the image is displayed.

3.14 Map Scale - Definition and Display

This section describes the significance of map scale, and the routine that draws the

scale bar for a image map displayed in the graphics window. A map is dimensionally related to the real world through scale which, by definition, is a ratio of the distance on the map to distance it represents on the surface of the earth [Rob78]. It can be represented in various ways: by a numerical ratio (also referred to as the representative fraction), in which one unit of measurement on the map is given the equivalent number of units on the ground. (e.g. 1: 10,000); by a statement of equivalent distance on the map and on the ground, commonly using units of measurement suitable to their relations (such as 1 inch represents 4 miles); or by a scale bar, which graphically indicates distances on the ground at their map scale.

In SISP, the scale is the ratio between the display coordinates and the distance it represents on the surface of the earth. The scale bar, shown in Figure 3-17, is displayed in the margin at the bottom of the display window. The system of measurement in which the scale is computed, is displayed below the scale bar. The ratio displayed below the scale bar is an alternative form of representing the scale. It is the ratio of one display coordinate, to the distance it represents on the surface of the earth. Thus unlike the conventional scale ratio, the quantities are not in the same units of measurement.

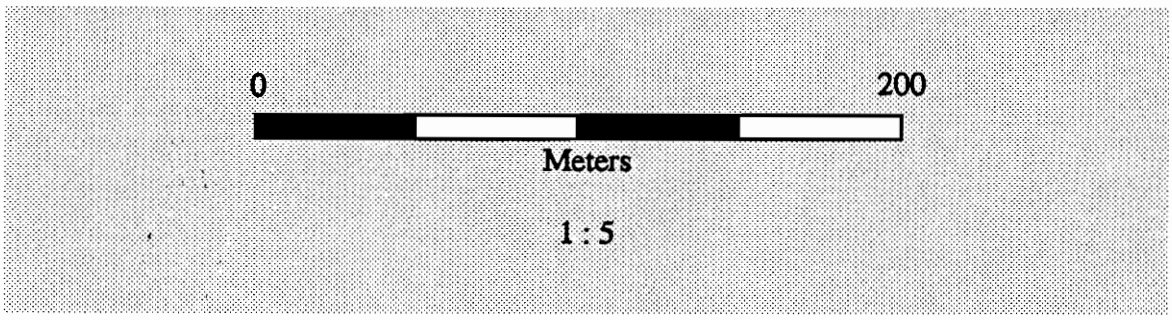

Figure 3-17. Map scale in SISP. It is represented by a scale bar and a representative fraction.

The scale factor that has to be displayed, is computed along with the coordinate transformation factors, which are computed when the image is displayed. This factor, to convert between the number of data points that constitute the image and the display coor-

dinates, is computed such that displayed image is isotropic. In an isotropic image, the unit distances along the horizontal and vertical axes are equal. When the displayed image is zoomed into, the scale displayed reflects this change in the resolution at which the image is displayed.

The scale displayed should be interpreted with caution, since by the very nature of the necessary transformation from the sphere to the plane, the scale on a map must vary from place to place and will commonly vary even in different directions at a point. The scale can however be used to obtain a close estimate of distances, on the map displayed in the graphics window.

3.15 Overlays on displayed raster images

This function is used to overlay the raster image displayed, with other types of data. Currently overlays are done for displaying the buildings (2-D representation) in a given geographical area, and for displaying icons representing user-defined objects, like antennas etc. A detailed discussion of this operation is provided in Chapter 4.

Overlaying building data and other user-defined map features in a given geographical area is of great importance. Radio wave propagation in an urban environment is influenced to a great extent by the presence of buildings, trees and other map features in the area. The path from the transmitter to the receiver is often not line-of-sight. There can be multiple paths to the receiver caused by rays that reflect off objects in the environment. The height of transmitters and receivers are also important while characterizing the propagation environment using the site specific models.

The user is provided with the option to overlay building data over displayed raster data. The building data is stored in a binary vector format. The format consists of a set of vectors to denote the outlines of each building. The coordinates of the vectors are provided in UTM coordinates. When overlaying the data, the coordinate transformation factors computed, prior to displaying the raster data, is applied for the overlay also. Similarly, only the regions of the vector file which are within the geographical area of the raster

image are displayed. When the user zooms into the displayed region, the building vectors are redrawn to reflect the changed zoom factor.

Besides overlaying building data, the user might also need to define objects in the environment, to determine its influence on propagation in the area. Additionally, the location of transmitters and receivers, need to be specified interactively. The user can, using the mouse pointer, specify the location of transmitters and receivers. An icon, is displayed at each location the user chooses. The icons are of dimensions 10x10 pixels, but they can be of different shapes and colors to denote different object classes. The user can associate a unique icon with each object type. A display of terrain elevation data overlaid with buildings and icons for transmitters and receivers in shown in Figure 3-18.

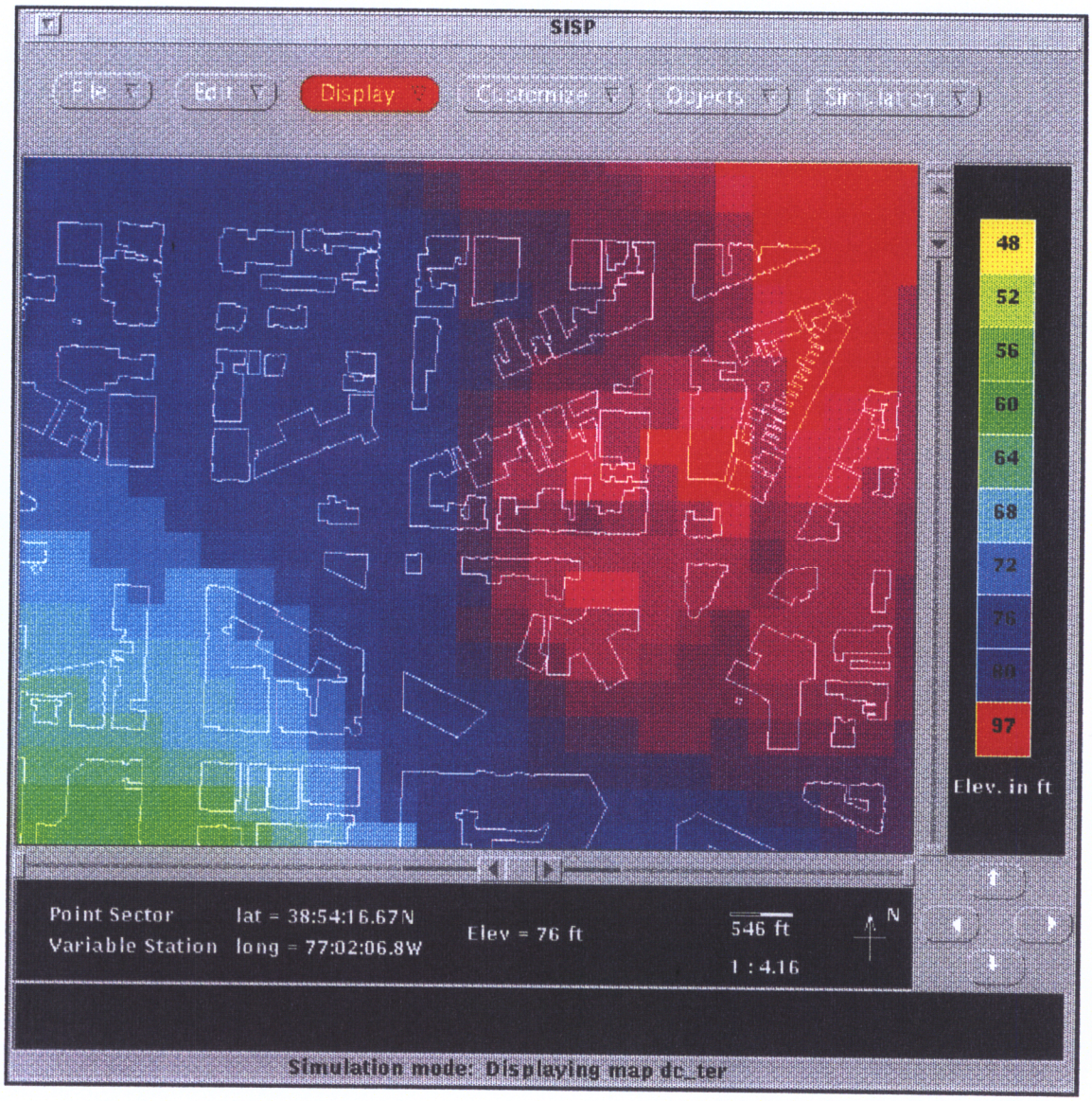

Figure 3-18. Display of raster data overlaid with vector data.

When the icon is displayed, in a location chosen by the user, the underlying raster image is saved. An image of dimension 10x10 pixels is saved using the Xlib function *XGetImage ()*. The image data is stored in a table, in the location indexed by the screen coordinates. This table contains an entry for all the antennas that have been located by the user during the current session. This is necessary since the user might, interactively, want to make changes to the locations, or delete them entirely. In these cases the raster image that was overlaid by the icon is restored. When the user deletes an antenna location, the screen coordinates of the location are used as the index to access the table, and retrieve the saved image. The saved image is restored in the window using the Xlib function *XPutImage ()*.

3.16 Creating icons for antenna types

This function is used to create icons for antenna types. These icons can be uniquely associated with each antenna type, and serve as an effective visual mechanism to distinguish between the different antenna types in a system. The icons are created from X bitmap (XBM) files, created by the user using a standard bitmap editor. A common bitmap editor is - bitmap - a tool which accompanies the standard X distribution.

The bitmap files are stored as simple C program fragments that can be compiled directly into programs. The format of a bitmap file is shown in Figure 3-19. The width and height of the bitmap as well as the hotspot, if specified are written as preprocessor directives at the start of the file. The image is then written out as a character array. Each character in the array contains 8 bits from one row of the image (rows are padded out at the end to a multiple of eight). The bitmap is saved in the row major order.

XBM FILE FORMAT

```
#define name_width 11
#define name_height 5
#define name_x_hot 5
#define name_y_hot 2
static char name_bits[] = {
        0x91, 0x04, 0xca, 0x06, 0x84,
        0x04, 0x8a, 0x04, 0x91, 0x04
    };
```

Figure 3-19. X bitmap file format. The file is stored as C program segment which can be included along with the application and compiled.

If the bitmap file is included along with SISP as a C program segment, it requires that all the icon bitmap files are included with SISP at compile time. Since the user might want the flexibility to create new icons, and associate them with antenna types at run-time, the above method was found to be unsuitable. The bitmap files are thus specified by the user at run-time.

The bitmap files are read and a pixmap is created using the Xlib function *XCreatePixmapFromBitmapData ()*. This function uses the background and foreground colors, defined for the window, to create the pixmap. The colors desired for the icon can be specified before the pixmap is created. The icon image is obtained from this pixmap, using the function *XGetImage ()*.

3.17 Zoom Operation

This operation is useful for displaying an image at a user-specified zoom factor. In

most displayed images the desired information has to be sought, and this process of looking for the detail, relevant to the query, is called visual search. Often the displayed data may not provide adequate information to the user. In order to find particular information or to extract a combination of elements to be understood, from the displayed image, the user might want to alter the display resolution. The main considerations in these operations are that the user should be able to specify the area to be zoomed in easily, and that the zoom operation should not be slow.

During the initialization process for a graphic window a pixmap is allocated, which is used for drawing raster images when the zoom factor is unity. A table is setup to store information about currently active pixmaps. A flag is used to indicate what the current zoom factor is, and the pixmap that is used depends on this flag.

When the user specifies a zoom factor greater than unity, the software checks the table to determine if the pixmap is already active. If the pixmap is not active, a pixmap of the required dimensions is allocated. The width and dimensions of the new pixmap is the zoom factor times the dimensions of the original pixmap. Once the pixmap has been allocated, the flag is set to indicate the current zoom factor.

The zoom factor is used to compute a new set of coordinate transformation factors for displaying the data. The data to be displayed and the color scheme to be used, was initialized when the original image was displayed. The color encoding scheme is used to assign the RGB values to the data values. The RGB values are then used to lookup the pixel value from the colormap. When the image is being drawn, the drawing operation is performed on the visible window and also on the currently active pixmap. If the user specifies a zoom factor which was used earlier, the image is not composed again, but is copied from the pixmap to the display window.

When the user specifies a zoom factor greater than unity, only a sub-region of the entire image is displayed in the window. The other regions have to be panned into, to view the entire image. The zoom operation draws to the visible window and to the pixmap that is currently active. Hence even if the display window displays only a sub-region of the

image, the image for the other regions is stored in the active pixmap.

When different regions of the map are to be displayed, the bounds of the region, specified by the location of the viewport and the dimensions of the window is calculated. This region is copied to the window using the Xlib function *XCopyArea ()*. This provides for a fast pan operation.

The scroll bars along the horizontal and vertical axis can be used to scroll the displayed image by small increments.

3.18 Saving raster images

The terrain data files, typically, contain data that represents the terrain elevation information of large geographic regions. Very often, the region of interest to the user is only a sub-region in this data file. The user can utilize the zoom operations to identify and display the sub-region of interest. Since this region might have to be used multiple times, the region needs to be saved for later use. This routine is used to save the terrain elevation information associated with the region of the displayed image, that has been selected by the user.

This routine is invoked when the user needs to save the specified region of interest. Since this routine saves the terrain elevation information, and not the pixel values that compose the displayed image, it is provided with the name of the terrain elevation file that is currently displayed. This routine saves the terrain elevation information as an terrain data file in the uncompressed format. The current propagation prediction modules utilize the data in this format. This format has been described in Section 3.5.1.

Routines have also been developed to import the uncompressed terrain data into the SISP database. The imported data is stored using run-length encoding. The format of this data has also been described in Section 3.5.1.

3.19 Setting color choices

The user can set the background color and the foreground color in the graphics context associated with the display window. The foreground color and the background colors of the graphics context are used for all the drawing operations in that window.

The values for the desired color are defined in a header file *colors.h*. The color values that are passed as the foreground and background color are used as indices to an array of user-defined RGB values. These RGB values are used to allocate the colorcells from the colormap installed for window. The Xlib function *XAllocColor ()* is used to allocate the colorcells from the colormap. If the routine is unable to allocate the desired foreground and background colors, it returns an error value rather than attempting to create a new colormap.

This chapter has introduced the SISP raster toolkit. It commenced with an introduction to geographical information systems and established the need for GIS functionality in site specific propagation prediction software. The capabilities of the raster toolkit that has been developed was presented and the functionality it provides was explained. The subsequent chapters describe the SISP vector library and provide a description of the various data formats used for measured and predicted data in SISP.

Chapter 4. SISP Vector Library

4.1 Overview

The previous chapter provided a detailed description of the SISP raster library. The raster function library provides an efficient interface for the storage, retrieval, and visualization of raster image data. Building and other map feature data in a geographical region are obtained in a vector format. The essential difference between the raster and vector formats is the technique used to represent the map features. Since the representation of information is fundamentally different, a library of functions has been developed to manipulate this data. Detailed descriptions of the SISP binary vector format, the vector library, and other related programs are provided in this chapter.

This chapter commences with an introduction to vector data. It describes the binary format in which SISP stores the vector data. The functionality of the routines used to visualize this data is also discussed. The data for buildings, in a given geographical area, is obtained as AutoCAD DXF (Data Interchange Format) files. This chapter specifies and provides a description of the format in which this data is obtained. A detailed discussion of the AutoCAD DXF format is provided in Appendix of this thesis. A detailed functional description of the program, that has been implemented, to import AutoCAD DXF data into the SISP vector database is also described here.

4.2 SISP Vector Data Format

This section contains a detailed discussion of the SISP vector data format. The format of these files determines the functionality of the SISP vector library. Vector data differs from raster data in its representation of map features, present in the environment, in the data file. In the vector format, the various features are stored using the arc-node representation.

Prior to the discussion of the arc-node representation, a description of all the terms involved, is in order. Arcs are non-intersecting curves that are stored as a set of coordinate

pairs. The arcs are also referred to as vectors for this reason. The two end-points of an arc are called nodes. Two consecutive nodes-*(x, y)* coordinate pairs-define an arc segment. The arcs either singly, or in combination with others form high level map features such as lines or area edges. Arcs that form linear features are called lines. Arcs that outline areas are called area edges or area lines.

As was discussed in the previous chapter, the SISP raster data consists of a data file and a set of support files. Similarly, the vector data consists of a data file and support files, with the same name, in different directories in the SISP database. The primary database directories are shown in Table 4-1.

Table 4-1. Database directories for vector data.

Directory	Function
dig	Binary vector file
dig_ascii	ASCII vector file
dig_att	Vector attribute file

The vector file is usually stored as a binary file in the directory *dig*. The ASCII version of this file is present in the directory *dig_ascii*. The attribute file is located in the *dig_att* directory, and contains the attributes for the map features in the data file. For instance, a vector map named dc.building would have the binary data stored in the file *dig/* dc.building, with the support files *dig_ascii/*dc.building and *dig_att/*dc.building.

A discussion of the SISP vector data format and the format of the support files is presented in the sub-sections that follow.

4.2.1 Vector Data File

The SISP vector file format is similar to the GRASS-4.0 vector file format. The vector data file consists of two distinct sections: the header section and the data section. The header section is mandatory, and is present at the start of the file. It provides information about the geographical region in which the map features are located, the map scale, and other information pertaining to the data set. Following the end of the header section is

the data section. This contains information corresponding to each map feature. An outline of the organization of the SISP vector data file is shown in Figure 4-1.

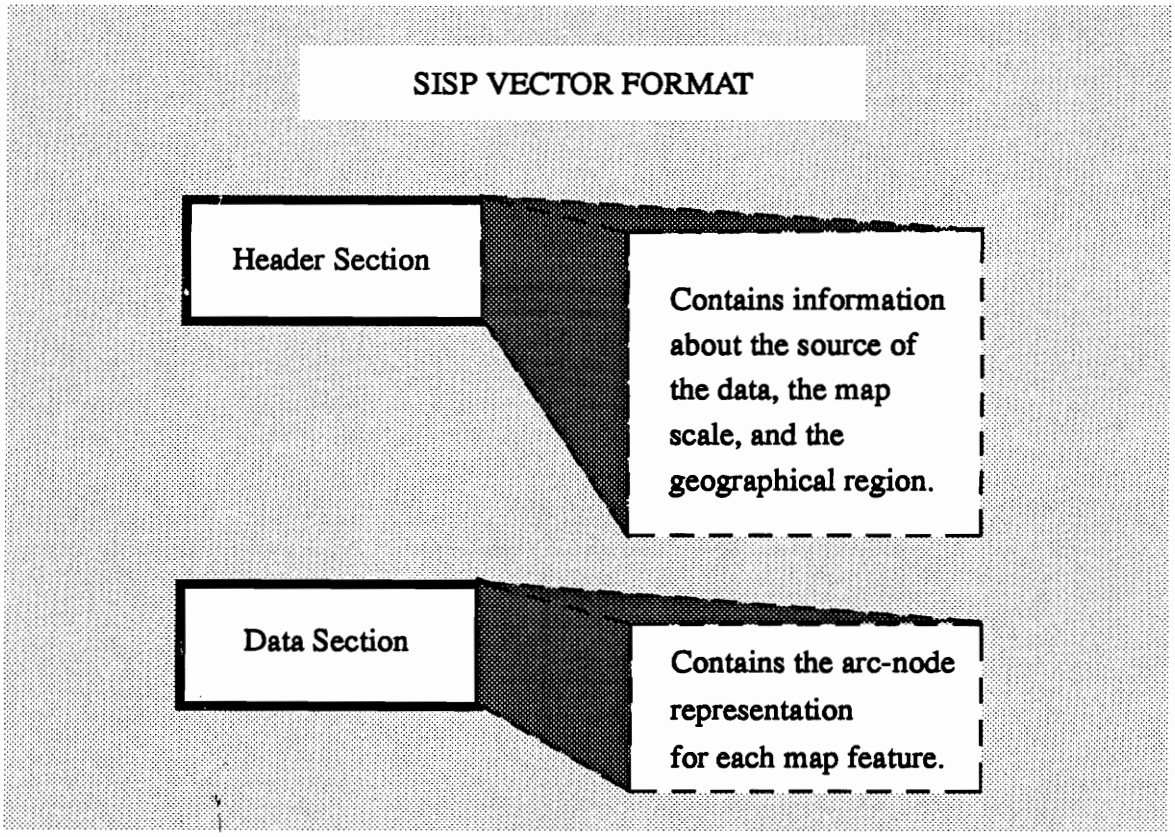

Figure 4-1. Outline of the organization of the SISP vector data file.

The format of the ASCII vector data file is presented below, and a segment of the data file is shown in Figure 4-2. The header section of the file and the arc-node representation of one map feature has been shown in this figure. The binary format for this data is described at the end of this section. SISP can import vector data and store the ASCII and binary version in its database.

VECTOR FILE FORMAT

```
ORGANIZATION:  MPRG
DIGIT DATE:    Jan 19, 1994
DIGIT NAME:    T. P. Subramanian
MAP NAME:      dc.building
MAP DATE:      Jan 19, 1993
OTHER INFO:    Building map of Washington DC
MAP SCALE:     24000
ZONE:          18
WEST EDGE:     322735
EAST EDGE:     324025
NORTH EDGE:    4308558.3
SOUTH EDGE:    4307418.3
MAP THRESH:    1.0
VERTI:
L5
 4307662.7 322932.6
 4307661.6 322975.7
 4307665.5 322975.6
 4307665.3 322983.1
 4307654.8 322982.4
```

Header Section

Data Section

Figure 4-2. Segment of a SISP ASCII vector data file. The figure shows the header section of the file and the arc-node representation of one map feature.

- **Header Section**

The header section contains, besides other information, historical information pertaining to the data set, a description of the data, and its geographical location. The header

section of the vector file contains a total of fourteen fields, as shown in Table 4-2.

The format of these fields is discussed in this paragraph. The labels, of the various fields in the header, start in column 1 and continue through to column 14. The labels are all uppercase, left-justified, and end with a colon. All the labels are blank padded to column 14 and the contents of these field start in column 15. All the fields shown in Table 4-2 are mandatory and must occur in the order shown in the table.

Table 4-2. Contents of the header section in the SISP vector data file

Label	Description
ORGANIZATION	Organization that digitized the data
DIGIT DATE	Date the data was digitized
DIGIT NAME	Person who digitized the data
MAP NAME	Name of the original source map
MAP DATE	Date of the original source map
OTHER INFO	Comments about the data file
MAP SCALE	Scale of the original source map
ZONE	UTM zone of the map
WEST EDGE	Coordinates of the western edge of the region
EAST EDGE	Coordinates of the eastern edge of the region
SOUTH EDGE	Coordinates of the southern edge of the region
NORTH EDGE	Coordinates of the northern edge of the region
MAP THRESH	Resolution at which the map was digitized
VERTI	No data. Marks the end of the header.

The significance of the various fields in the header section of the data file is explained in this paragraph. The *ORGANIZATION* field contains the name of the organization from which the file was obtained. The *DIGIT NAME* and *DIGIT DATE* fields give the

name of the person who digitized the data and the date when it was digitized. The *OTHER INFO* field contains information pertaining to the data in the file, like the title for the data or information about the source of the data. The *MAP SCALE* field contains the map scale of the original map. The *ZONE* represents the UTM zone of the geographical area for which the data has been obtained. A detailed description of the UTM coordinate system was provided in Section 3.6 of Chapter 3. The *WEST EDGE, EAST EDGE, SOUTH EDGE,* and *NORTH EDGE* fields specify the UTM coordinates of the geographical region, for which the data was obtained. The *MAP THRESHOLD* field contains the resolution at which the data was digitized. The *VERTI* field does not contain data but indicates the end of the header section of the file. Default values are provided for the fields which are irrelevant to a particular data file, or for which no data is available.

- **Data Section**

Following the header section in the vector file, is the data section. It consists of the map features which are specified as sets of vectors. Each set of vectors consists of a feature descriptor and the number of vertices, followed by a series of coordinate pairs for the vertices. The feature descriptor specifies the type of map feature, and can have either of two values: "A" to denote an areal feature and "L" to denote a linear feature. The number of vertices specifies the number of vectors that the map feature comprises, and is followed by the coordinate pairs. A segment of the data section of the file is shown in Figure 4-3. The map feature shown in the figure is an areal feature comprising 5 vectors. The feature descriptor is the character A, in the first column. The number of coordinate pairs are separated from the descriptor by at least one space. The vertex coordinate entries start with a space in the first column, and have at least one space between the two values. The coordinates shown in the figure are UTM coordinates and are stored in the file as northing and easting values, in that order.

The binary format of the vector data file contains all the information in the ASCII version of the file. The header section is present at the start of the file, but the labels identifying each field in the header section is omitted. The data for each feature in the binary

format consists of a feature descriptor, the number of vectors that compose the feature and the coordinates for the end points of these vectors. However, the format in which the coordinates are stored in the file differs from the format in the ASCII file. The easting values of the nodes of all the vectors, for a map feature, are stored in a contiguous block in the file, which is followed by a block of northing values.

ARC-NODE REPRESENTATION	
A 5	
4307662.745	322932.658
4307661.604	322975.705
4307665.555	322975.664
4307665.339	322983.184
4307654.847	322982.408

Figure 4-3. A segment of the data section of the vector data file.

4.2.2 Vector Attribute File

The vector data file contains the data that specifies the extents and contour of the map feature. Additionally, each map feature can have an attribute assigned to it. The attribute value for each map feature is not stored in the data file located in the *dig* directory or the *dig_ascii* directory, but in an attribute file located in the *dig_att* directory. The attribute file is an ASCII file that contains an entry for each of the map features in the data file. The format of a typical entry in the attribute file is shown in Figure 4-4. Each entry consists of a descriptor for the map feature, a coordinate marker identifying the map feature, and the value of the attribute.

MAP FEATURE ATTRIBUTES			
A	389668.32	4433900.99	106.5
L	388142.16	4434446.65	65.6

Figure 4-4. Attribute file format. The figure shows 2 entries from an attribute file. The values 106.5 and 65.6 denote the attribute values for the features.

The feature descriptor field contains one of the characters "A" or "L". As in the data file, these characters denote areal and linear features respectively. The coordinate marker is used to identify the feature with which the attribute is associated. The coordinates are in UTM and are stored as easting and northing values, in that order. This is different from the format used for coordinate entries in the data file (northing and easting). The marker must be located such that it can be used to uniquely identify the related map feature. In particular, a coordinate marker for an areal feature should be within the bounds of the feature. A coordinate marker for a line feature should be on or close to the corresponding line feature, and should be closer to its related line than any other. Additionally it should not be located at a node. If multiple entries identify a map feature, the last one encountered when the attribute file is read sequentially from the start, will be used. A map feature that has no entry in this file, is considered to be unlabeled.

4.3 Vector Topology Rules

This section presents the rules which need to be applied when a vector data file is created for importing it into the SISP database.

1. The arcs in the data file should not cross each other. Arcs which would cross each

other must be split at their intersection to form distinct vectors.

2. Arcs which share nodes must end at exactly the same points. This is important since nodes are not explicitly represented in the data section of the data file, but implicitly as endpoints of arcs.
3. Common boundaries between map features should occur only once in the data file.
4. The areal features must be explicitly closed. This implies that it must be possible to complete each area by following the edges connected by common nodes.
5. The areal features and linear features should be positioned in separate map layers, or data files. However if the areal features and linear features must appear in the same layer, then the common boundaries must be digitized only once. An area edge that is also a line, should be digitized as an area edge to complete the areal feature. The areal feature should be labeled as an area by providing a value "A" for the feature descriptor, in the data file. Additionally, the common boundary arc should be labeled as a linear feature by providing a value "L" for the feature descriptor, in the data file.

Applying the general rules presented above to any source data file that is created, minimizes the possibility of error when this data is imported into the SISP database. The guidelines apply irrespective of the source of the data file. For instance, these rules hold good for files generated by digitizing paper maps and also to files that are obtained from CAD software like AutoCAD.

4.4 Displaying Vector Data

This section describes the functional details of the routines that the SISP vector library comprises, and which aid in the visualization of vector data. The chief objectives of these routines are stated below

- Verify the existence and check access permissions of the vector data file and the attribute file.

- Open the vector data file in the read-only mode and initialize the data structures which store the file information.
- Check if a geographical region has already been associated with the graphics window.
- Allocate memory to read the coordinate pairs for each map feature.
- Read the data corresponding to each map feature from the data file.
- Generate a bounding box enclosing all sides of the map feature.
- Determine if the bounding box is contained, either entirely or partially, within the geographical region associated with the display window.
- Perform coordinate transformations on each coordinate pair of the feature.
- Display the feature and draw to the pixmap using the Xlib drawing primitives.

As was discussed in Section 4.2, the vector data is essentially composed of the data file and support files in the database. The routines that display the vector data read the binary vector data stored in the *dig* directory in the database. The attribute file is not directly utilized by any of the display routines. However, the information stored in this file is extremely important. For instance, for propagation models which employ 3-dimensional ray tracing techniques, height attribute of the buildings in the area is essential. Hence, the display routines check the existence of the attribute file and return an error if the file does not exist. The vector data file in the *dig* directory is opened for reading. The header section of the file is read and copied to the appropriate data structures in memory.

After the data file has been opened, the routine determines whether a geographical region has been assigned to the display window. For instance, if a terrain map had been displayed in the window, prior to the vector display operation, then the geographical region associated with the raster data file, would be the geographical region assigned to the display window. The bounds of this geographical region automatically becomes the bounds for the vector file, and only the map features that lie within this region are dis-

played. Moreover, the coordinate transformation factors that were computed during the previous graphic operation are used by this routine. If the display window does not have a geographical region associated with it, the geographical region specified in the header section of the vector file is assigned to the display window. In this case, the coordinate transformation factors to be used for the display operation are computed.

Once the coordinate transformation factors have been computed, the routine reads the data file to perform the display operation. The data for each map feature consists of a feature descriptor, the number of coordinate pairs that the feature comprises, and the coordinates for the vertices. As was described in 4.2.1, the coordinates in the binary vector data file are stored as separate blocks of easting values and northing values. The routine allocates memory to read in a large number of coordinate pairs. If the number of coordinate pairs that compose a map feature exceeds the block of memory that is allocated, additional memory is allocated by the routine. This technique improves the performance of the drawing routines, since it does not require memory to be allocated each time a feature is displayed. Additionally, it prevents memory fragmentation which can occur as a result of allocating and freeing small blocks of memory.

Subsequent to allocating memory, the routine iteratively reads in the data for each map feature and displays the feature. At first, the routine reads the number of coordinate pairs for each map feature and determines if the amount of memory allocated is sufficient for storing the coordinate data. A larger block of data is allocated if the memory is insufficient. The easting and northing values are read from the data file and copied to this block of memory.

The coordinate data for each map feature is used to determine if the map feature is located in the geographical region, in effect for the display window. The bounding box technique is used to determine this information. For each map feature a bounding box is computed such that it encompasses all the vectors that the map feature comprises. The coordinates of the bounding box are checked with the coordinates of the geographical region to determine if the map feature needs to be displayed. If the entire bounding box

lies outside the geographical region, the map feature is not displayed. If a portion of the bounding box lies outside the geographical region, the map feature is displayed, clipped to the extent of the geographical region of the display window.

After the bounding box test has been performed, screen coordinates are obtained by applying the coordinate transformation factors. The map feature is displayed by passing the display coordinates as arguments to the Xlib function *XDrawLine()*. The color of the map feature displayed by iterative calls to this drawing primitive, depends on the foreground color that is in effect for the display widow. The map features are also drawn to the active pixmap, to improve the speed of redraw operations.

The sequential steps in the display of the vector data are represented in a conceptual graphics pipeline, shown in Figure 4-5. The key points of the pipeline are described below.

- The input section represents the routines that read in the data from the disk into the appropriate data structures in memory.
- The modeling transformation section represents the process of setting the geographical region and computing the coordinate transformation factors.
- The rectangular clipping section represents the bounding box test to determine if the map feature is to be displayed or not
- The viewing transformation section represents utilizing coordinate transformation factors to the data, to obtain display coordinates.
- The display section represents Xlib drawing primitives that are used to display the map features in the display window.

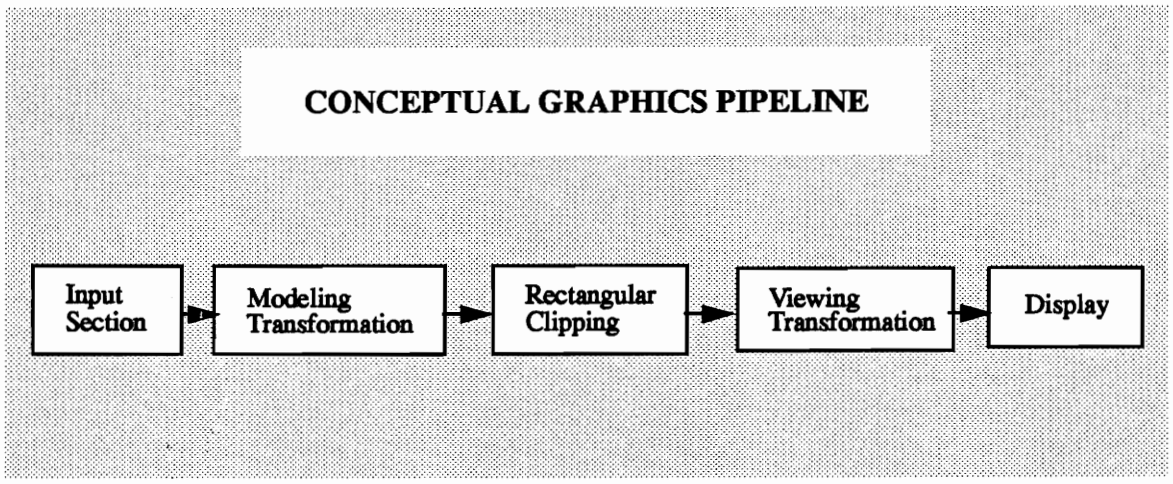

Figure 4-5. Conceptual graphics pipeline model for vector data processing.

Thus, this routine provides for the visualization of vector data stored in the SISP binary format. Since the routines that read the data from the vector file, are in discrete modules from the routines that perform the display operations, the same display routines, with minimal modifications, can be re-used for the visualization of data in other formats.

4.5 Acquisition Of Building Data

The accuracy of site specific propagation prediction models, depends on the precision with which the various map features are represented. The propagation models that are based on ray tracing techniques require precise information about the contours of buildings and their heights. While USGS maps serve as a reliable source of terrain data, building data can be obtained in multitudinous formats from several sources.

A common technique of acquiring building data is from aerial photographs, by using photogrammetric techniques. The first step in this process is making aerial photographs of the geographical area for which the building information is required. Photogrammetric techniques such as stereoscopy are then used to obtain the necessary information from these aerial photographs. Stereoscopy is the technique by which x, y, and z spatial coordinates are obtained from stereo photographs [Wel89], [Wel90]. This

information is digitized and imported into standard CAD packages like AutoCAD.

AutoCAD is a computer-aided design and drafting software which is very commonly used for civil engineering applications. The drawings in AutoCAD are stored in a proprietary binary format. The map features are represented as AutoCAD objects. These objects can have several different internal representations, each differing in the technique used to draw them. A standard technique to represent the attribute information associated with these objects is by using its properties. Some commonly used properties are line thickness, line type, and line color. Other methods such as using text to explicitly denote the attribute are also used.

Although AutoCAD files are stored in a compact binary format, it is not the recommended format for data interchange, the reasons being: it is likely to change significantly as new features are added and it is not documented explicitly. The Drawing Interchange Format (DXF) [Acad90] has been defined to serve as the format for the exchange of drawings between different software packages. The advantage of the DXF format is that, all implementations of AutoCAD can export and import data in this format. An overview of the structure of these files is provided in Section 4.6

The DXF files serve as the primary format in which building data is obtained for SISP. As was discussed in the preceding paragraph, there are several different techniques for representing map features. In AutoCAD DXF files for use with SISP, it has been established that two dimensional polylines be used to represent the contours of the buildings, and the thicknesses of these polylines to represent the heights of the buildings. There is only one value of height available for the entire building, and this represents the height of the building above mean sea level. The data currently being used has been supplied by Photogrammetry Data Services, Inc. in Northern Virginia.

4.6 AutoCAD DXF File Format

The DXF files contain an exhaustive description of a drawing. The file contains information about the entities and information essential for viewing the drawing in

AutoCAD. An overview of the structure of DXF files is shown in Figure 4-6.

It can be observed from Figure 4-6 that the DXF files are organized into five distinct sections. These sections are the header section, the tables section, the blocks section, the entities section, and the end of file section.

- The header section contains general information about the file necessary for displaying the drawing.
- The tables section contains information about the different tables used to store information about the drawing such as linetype table, layer table, etc.
- The blocks section contains information about blocks, which are sets of objects grouped together.
- The entities section contains information about the individual objects in the file. This information includes the layer in which the object is located, coordinates to represent the contour of the object, and other properties.
- The end of file section denotes the end of the DXF file and does not contain any data.

Numeric codes are used to represent the information about the objects in the drawings. Separate codes are used to denote the properties of objects, such as line thickness etc. A segment of an AutoCAD DXF file is shown in Figure 4-7. The detailed description of the group codes and the various flags that used to represent entity information is provided in Appendix of this thesis.

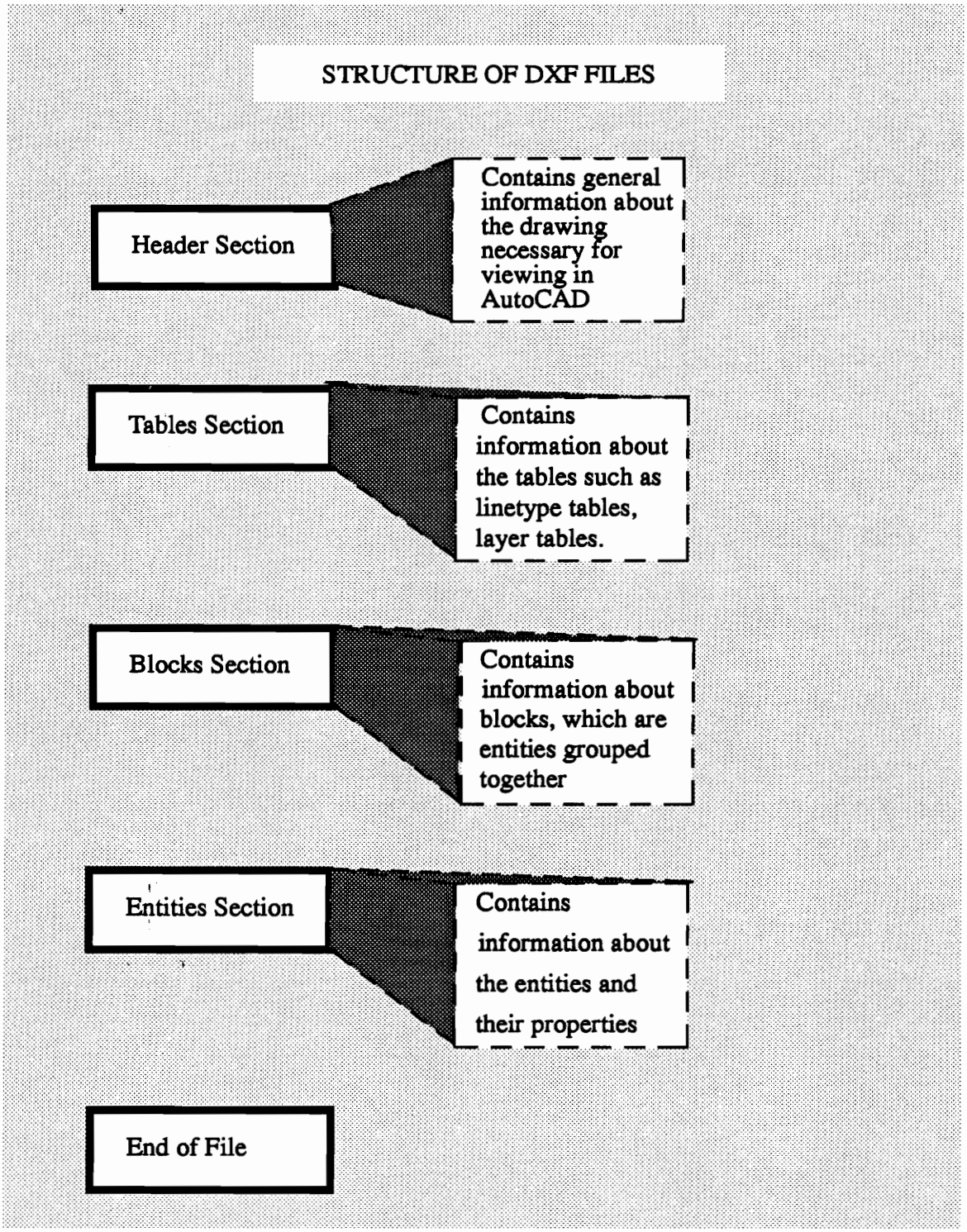

Figure 4-6. Overview of the structure of DXF files.

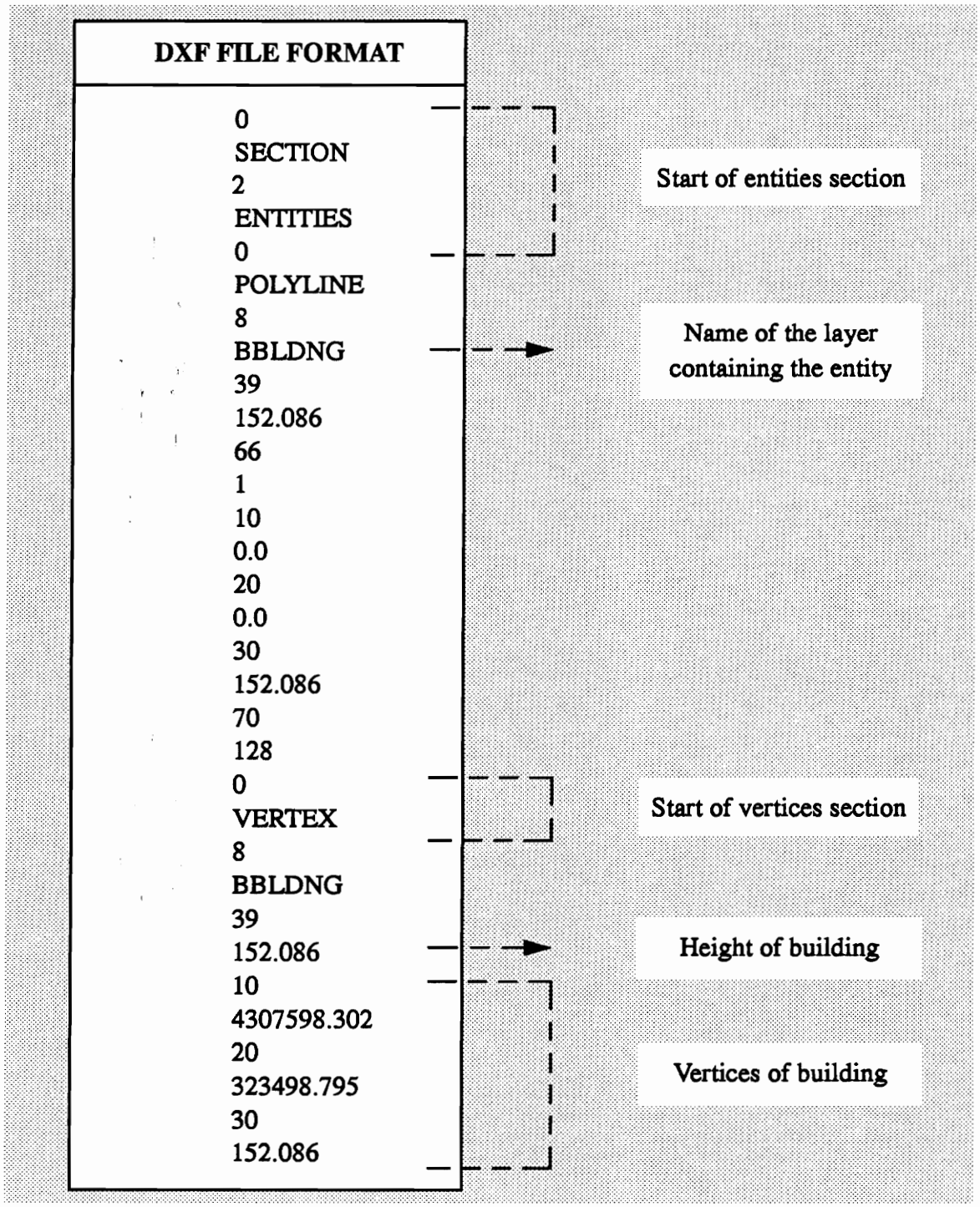

Figure 4-7. Segment of an AutoCAD DXF file. The figure shows the representation of an object and the layer in which it occurs.

4.7 Conversion From DXF to SISP Binary Vector Format

This section describes the program *DxfConv*, that performs the conversion from AutoCAD DXF files to the SISP binary vector format. As was discussed in Section 4.5, building data can be obtained in several different formats. The advantage of using a standard format for the data such as the DXF format, was also emphasized. Thus, the AutoCAD DXF files serve as the primary format for building data for SISP. The objectives of the conversion program are listed below.

- Verify access permissions to the AutoCAD DXF file which is provided as a command line parameter.
- Obtain, from the user, the name of the layer in the DXF file from which the objects have to be imported.
- Ignore sections of the file which do not provide useful information, and scan for the start of the entities section.
- Read each entity and determine if they are located in the "import" layer.
- Create and allocate appropriate data structures to which the entities are copied.
- Create a temporary entities file in the SISP ASCII vector format, conforming to the vector topology rules suggested in Section 4.3.
- Invoke routines that perform the conversion from the SISP ASCII vector format to the binary format.

The conversion program accepts the name of the DXF file to be processed and the name of the vector file to be created, as command line parameters. The program can be invoked as:

DxfConv dxf_filename vector_filename

The program prompts the user to specify the name of the layer from which the objects have to be imported. As was discussed in Section 4.6, the AutoCAD drawings usu-

ally contain several different layers. The layers contain objects that have been grouped on the basis of a set of norms. For instance, the different layers in a typical building data file could contain: buildings, the streets and sidewalks, the foliage etc. There are no restrictions on the number of layers that can be present in a file, or on the number of objects in each layer. The objects in different layers of the DXF file need to be stored as separate data files in the SISP database. Subsequent to obtaining the "import" layer from the user, the program verifies access permissions of the DXF file, specified in the command line. The DXF file is opened in the read-only mode. Furthermore, a temporary file to store the imported objects and an attribute file to store the heights of the buildings are created.

After the files have been successfully opened, the program reads the DXF file to obtain information of objects in the "import" layer. Since the program accesses the file sequentially, the various sections of the file, discussed in Section 4.6, are encountered. The first section in the file, the header section, contains information useful for viewing the drawings in AutoCAD. The following section is the tables section which contains tables such as the linetype table, the layer table etc. Since the data in the first two sections of the file is irrelevant for the conversion process, it is ignored by the program. The subsequent blocks section contains information about objects in the layer, which have been grouped using the AutoCAD *blocks* command. However, the objects that occur in this section are also present in the entities section of the file, and thus the program ignores this section of the file also.

From the discussion in the preceding paragraph, it is evident that the program does not interpret the data in the file until the start of the entities section. The entities section is identified by the presence of the group code 2, followed by the keyword *SECTION*, followed by the keyword *ENTITIES*, in that order. After the start of the entities section, the program reads each object and performs the conversion until the end of file section is reached. The following paragraphs essentially describe this process, interspersed with descriptions of the representation of objects in the file.

The format in which a polyline entity is represented in the file is shown in Figure

4-8. The numeric code 0 is used to denote the start of a entity, table, or file separator. The keyword *POLYLINES* denotes the AutoCAD drawing primitive that has been used to represent the object. The code 8 indicates that the next line in the file contains the name of the layer. Thus, in the figure, *BBLDNG* is the layer in which the objects are drawn. The code 39 denotes the thickness of the polyline used to draw the object and the code 66 indicates that the vertices that compose the object follow its description. A detailed description of the relevant numeric codes is provided in Appendix of this thesis.

DESCRIPTION OF A POLYLINE ENTITY
0 POLYLINE 8 BBLDNG 39 152.086 66 1 10 0.0 20 0.0 30 152.086 70 128

Figure 4-8. Description of a polyline entity in DXF files. The description contains information about the type of primitive used to create the entity, the layer in which the entity occurs, and the line thickness used to draw the entity.

For each object encountered, the program determines the layer in which it is located. The vertices of the object are read, only if the object is located in the layer specified by the user. Before reading the objects from the file, the program allocates a large block of memory to store the coordinates of the vertices of each object. For each object read, the program determines if the memory allocated is sufficient to store the vertices. If the memory allocated is insufficient a larger block of memory is allocated. Thus memory is not allocated and freed every time an entity is read. This technique has the advantage of preventing memory fragmentation, which can occur if memory is allocated repeatedly from the system in small quantities.

From the format of the entity description, it is evident that it does not contain any information about the number of vertices that the object is comprised of. The data for the vertices of the object is present at the end of the entity description. The vertex coordinates are read by the program and copied to appropriate data structures in memory. The format in which the vertices are present in the DXF file is shown in Figure 4-9.

DESCRIPTION OF A VERTEX
0 VERTEX 8 BBLDNG 39 152.086 10 323498.795 20 4307598.302 30 152.086

Figure 4-9. Description of a vertex entity. The description of the vertex, in a DXF file, contains information about the layer which contains the vertex, and the values of the x, y, z coordinates.

The line that follows the group code 8 denotes the name of the layer in which the vertex is located. The group codes 10, 20, and 30 denote the X, Y, and Z coordinates of each vertex. The Z coordinate denotes the height of the building represented by the entity. The end of the entity is denoted by the presence of the group code 0, followed by the key-word *SEQEND*. This keyword marks the end of the preceding entity and the start of a new one.

The entity description is written to the temporary file in the SISP ASCII vector data format, described in Section 4.2.1. The building is represented as an areal feature in the vector file by providing a value "A" for the feature descriptor. The number of vertices

comprising the entity and the coordinates of these vertices are also written to the vector file. The third coordinate of each vertex, as stated earlier, represents the attribute information for the map feature. Since the Z coordinate for all the vertices of the object are the same, only one attribute is written to the attribute file for each object imported. Once the end of file section has been reached, the program invokes the function that convert the ASCII vector file to the binary format. The program terminates when a binary file containing all the objects in the "import" layer has been created.

Several DXF files, currently being used by the propagation prediction programs, contain objects which have been drawn using the AutoCAD *PFACE* command. To be able to display such files from within SISP, they have to be converted to the SISP binary format. This program was thus extended to convert DXF files, containing buildings represented using polyfaces, to the SISP binary format. For buildings drawn using polyfaces, the *POLYLINE* entity is used to represent the contours of the buildings. However, the heights of the buildings are represented by the heights of the polyfaces and not by the thicknesses of the polylines used to draw these objects. The height of the polyfaces are obtained from a difference of their Z coordinates.

The program thus generates two files: the binary vector file, specified in the command line, in the directory *dig* and an attribute file, with the same name as the vector file, in the directory *dig_att*. This program can be run independent of SISP to perform the conversion.

4.8 SISP Plotting Library

This section provides an overview of the SISP plotting library. The primary function of the plotting library is to display measured and predicted data and polar plots of E-plane and H-plane sections of antenna patterns. This library comprises functions which manipulate data that are passed as parameters to the plotting functions. Since they do not read data files conforming to a particular format, they can be used for several purposes.

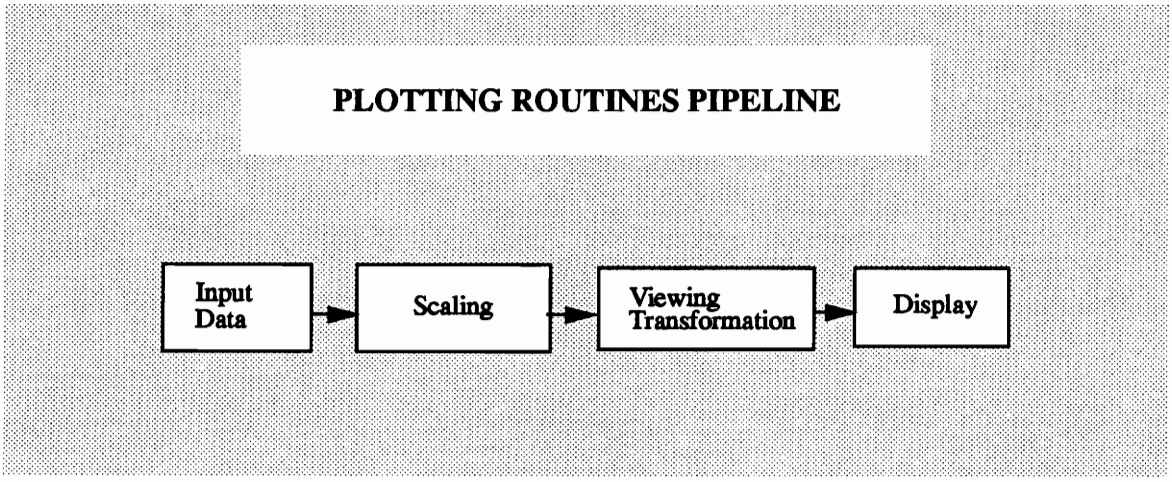

Figure 4-10. Conceptual pipeline for the plotting library routines.

The conceptual pipeline for the plotting library routines is shown in Figure 4-10. The key points of the pipeline are described below.

- The input data section represents the parsing routines that supply data to the plotting routines.
- The scaling section represents the routines that determine the limits within which the data is displayed. Moreover, it determines the number of intervals in which the data needs to be displayed.
- The viewing transformation section determines the dimensions of the plotting area, and computes the display coordinates for the data.
- The display section represents Xlib drawing primitives that are used to display the plots in the display window.

This chapter has described the SISP vector library. A detailed discussion of the vector file formats was provided. The functionality of the program that is used for the visualization of vector data was explained. The importance of having a standard data format to acquire building data was emphasized, and a standard format for importing building data into the SISP database was established. AutoCAD DXF files in which building

contours are represented by two dimensional polylines, and building heights by thicknesses of these polylines, constitutes this format. The filter program-*DxfConv*-to convert building data from the DXF file format to the SISP binary vector format was described. This chapter also included an overview of the plotting library that was developed to display measured and predicted data and two dimensional plots of the antenna patterns. A brief description of the DXF file format is provided in Appendix A.

Chapter 5. SISP Data Formats

5.1 Overview

The preceding chapters have discussed the programming libraries that have been developed to manipulate raster and vector data. The terrain data is stored in the SISP raster data format. The building data and other map features are stored in the SISP binary vector format. SISP uses two other data formats to import and store RF measurement data and the results generated by the various propagation prediction models. This chapter presents a discussion of these data formats[1].

This chapter describes the SISP ASCII format that is used to import field measurement data into the SISP database. It introduces a few of the data formats that are commonly used to represent technical data. It also presents the various issues involved in evaluating the suitability of a particular data format. The Hierarchical Data Format (HDF), developed by the National Center for Supercomputing Applications (NCSA), has been adopted as the binary format in which measured and predicted data is stored in the SISP database. This chapter discusses HDF and the various types of data organization that the software uses. A discussion of the programs that perform the conversion between the SISP ASCII format and HDF is also included in this chapter.

5.2 Specifications For Data Formats

This section lists some of the important specifications of data formats. These were considered when evaluating the various data formats for use in SISP.

- The data format should be machine independent and portable across different hardware platforms.
- The format should be self-describing such that all the information regarding the

1. In the sections that follow any reference to data implies both measured and predicted data, unless explicitly specified otherwise.

data set can be extracted from the data file.

- It should be capable of storing a variety of data types and different data organizations.
- The format should support the annotation of data.
- It should be widely used and be available as public domain software or in a proprietary format.
- It should provide the facility to store multiple data sets within the same file, with some description for each data set.
- The data format should be subroutine library based, in specifying the way files are written to disk, since these formats tend to provide better programming support for reading and writing files. On the other hand, disk-based formats specify the representation of the file on disk and are more portable.

5.3 Data Formats - An introduction

There are several data formats in use for storing technical data. This section introduces a few of the commonly used data formats [For92]. These descriptions are intended to provide a brief account of the various formats and are not a comprehensive discussion of their capabilities.

- **CDF (Common Data Format)**

CDF is a widely used data format standard developed by NASA for storing multidimensional gridded scientific data with user defined annotations. CDF stores the data in XDR (eXternal Data Representation) format, which is a proprietary standard for storing data in a portable form. CDF is primarily a subroutine library based format. The ability to annotate dimensions in CDF is limited.

- **DXF (Data EXchange Format)**

DXF is widely used in CAD applications to store polygonal data. Each data ele-

ment in DXF consists of two lines. The first line consists of a numeric code and the second line contains the actual data. DXF was created for AutoCAD. A detailed discussion of the DXF file format is provided in the appendix of this thesis.

- **FITS (Flexible Image Transport System)**

FITS is primarily used for storing astronomical data. FITS was originally designed to store images in a machine independent format. It has since been extended to store numerical data. This format has been endorsed by NASA's Science Data Systems Standards Office and the International Astronomy Union.

- **netCDF (net Common Data Format)**

netCDF is an extension of CDF developed by the National Center for Atmospheric Research (NCAR) and Unidata. netCDF stores multidimensional gridded data and contains information on dimensions, variables, and attributes. netCDF uses XDR for storing the data.

- **DLIS (Digital Log Interchange Standard)**

DLIS, also known as RP66 (Recommended Practice 66), is a data format standard created by the American Petroleum Institute (API) for geophysical data. It is used to store multidimensional arrays of various data types including floating point numbers, integers, and complex numbers. DLIS is a self describing format which supports the use of annotations such as name, units etc., of the dimensions.

- **PDS (Planetary Data System)**

PDS was developed by NASA's Jet Propulsion Laboratory (JPL) to handle data obtained from various planetary exploration missions. PDS generally stores image data in 2D matrix form as 8-bit unsigned integers. PDS also supports column data, spectral data, and 3D matrix data. PDS is machine independent and is intended to be a flexible formatting standard.

- **HDF (Hierarchical Data Format)**

HDF is a binary data format developed by the National Center for Supercomputing

Applications (NCSA). A detailed discussion of HDF is provided in sections 5.6 and 5.7 of this chapter.

5.4 Support Types for Measured and Predicted Data

The support type refers to the organization of predicted and measured data, on a geographical basis, for the purpose of visualization. The support type is completely independent of the dimensionality of the data. The basic support types that have been identified are: point data, grid data, path data and track data.

Point data refers to any data set associated with a single geographical point. The actual data may be single dimensional, as in the case of local average received power; two-dimensional as in the case of a power delay profile; or three dimensional as in the case of a spectrum analyzer capture obtained over a period of time.

Grid data refers to any data set which is associated with a geographical bounding area. Data is available at uniformly spaced intervals over the grid. The data is completely specified by the coordinates of the region, and the number of rows and columns of data available. Predicted path loss data over a geographical area is an example of this support type.

Some of the data such as instantaneous received power and power delay profile are inherently "track data". Track data is specified by the start and end points of the track or, alternatively, by the start point and the direction along the length of the track. Track data is different from path data in that track data is always along a straight line. Moreover track data is typically over much shorter distances than path data.

5.5 SISP General Data Format

The SISP ASCII data format provides a standard mechanism to import field measurement data and predicted data into the SISP database. The chief objectives of this data format are listed below.

- Establish a standard format for importing RF measurement data and predicted data, for analysis and visualization, into the SISP database.
- Provide a broad framework, general enough to encompass elements from model RF measurement efforts.
- Provide a format that can be read and edited by users.

The format comprises of two distinct sections: the prologue and the data section. The organization of the General Data Format file is shown in Figure 5-1.

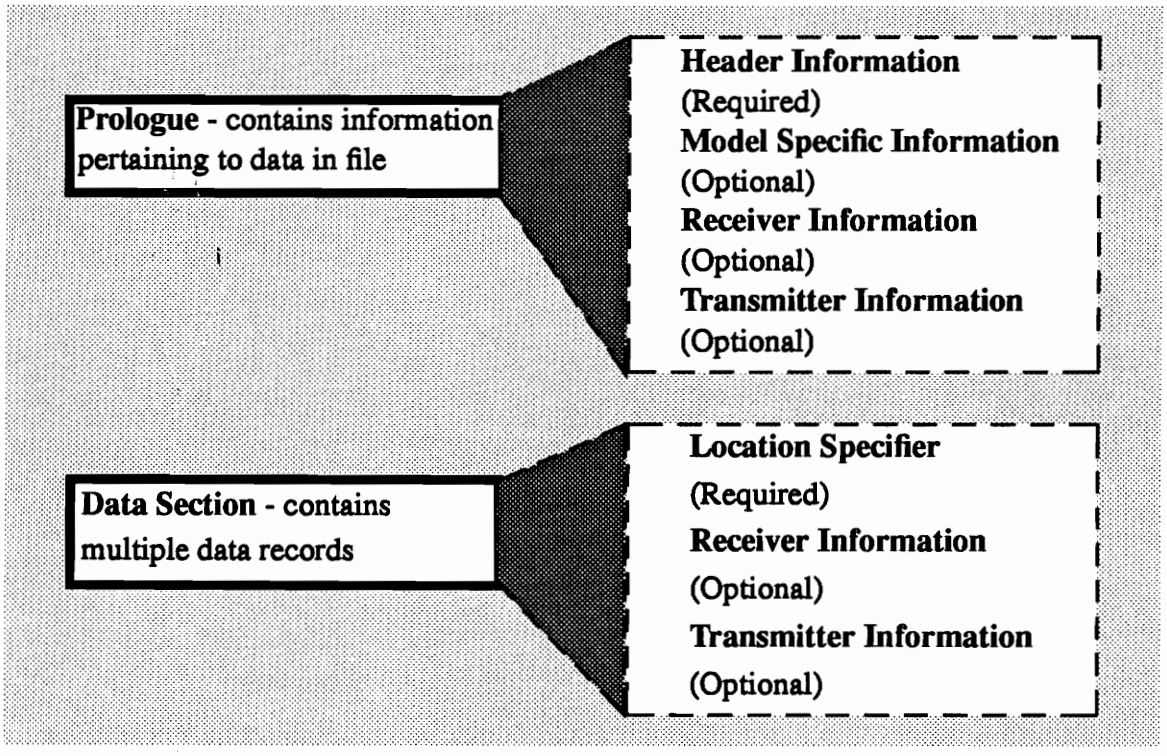

Figure 5-1. Organization of the SISP General Data Format file.

The header section specifies the source and nature of the data, and default parameters for the transmitter and the receiver. The data section contains the data records. The format of the SISP General Data Format is shown in Table 5-1, with the various fields and the possible values that the fields can contain. The following general guidelines apply to

the contents of the file. The keywords and string variables in the file are not case sensitive. The file name and the units fields are exceptions to this rule. Each keyword must start on a new line. Comments may be placed anywhere in the file by beginning them with open square bracket "[".

Table 5-1. The SISP General Data Format

SISP General Data Format	
[General_Parameters]	
Header_Begin	
Description:	*%s**
Date_Stamp:	*%s*
Time_Stamp:	*%s*
System:	*%s*
Variable_Station:	*Transmitter//Receiver//Variable*
Data_Type:	*%s*
Support_Type:	*Point // Track // Path // Grid // Variable*
Loc_Format:	*UTM // GPS // Referenced*
Reference_Pt_Loc_Format:	*UTM // GPS // N/A*
Reference_Point:	*%t*† *// N/A*
Building_Database:	*%s // Empty*
Terrain_Database:	*%s // Flat*
Num_Data_Recs:	*%ld*
Data_Origin:	*Measured // Predicted // Processed // Error*
Units:	*(Unit1 Unit2 ...)//Variable*
Header_End	
[Model Specific Information]	

Table 5-1. The SISP General Data Format

SISP General Data Format
Model_Specific_Information_Begin
Variable Parameters for each model
Model_Specific_Information_End
[Receiver_Antenna_Parameters]
Rx_Begin
Rx_Antenna_Height: *(%f [%s][%s]) // Variable*
Rx_Antenna_Type: *%s // Variable*
Rx_Antenna_Orientation: *(Direction: %f %f %f) // Omni // (Target: %t) // Variable*
Rx_Location: *%t // Variable*
Rx_End
[Transmitter_Antenna_Parameters]
Tx_Begin
Tx_Antenna_Height: *(%f [%s][%s]) // Variable*
Tx_Antenna_Type: *%s // Variable*
Tx_Antenna_Orientation: *(Direction: %f %f %f) // Omni // (Target: %t) // Variable*
Tx_Location: *%t // Variable*
Tx_Power: *%f [%s] // Variable*
Tx_Frequency: *%f [%s] // Variable*
Tx_End
[Data Record #1]
Data_Begin
Local_Rx_Begin

Table 5-1. The SISP General Data Format

SISP General Data Format	
Rx_Antenna_Height:	*(%f [%s][%s])*
Rx_Antenna_Type:	*%s*
Rx_Antenna_Orientation:	*(Direction: %f %f %f) // Omni // (Target: %t)*
Rx_Location:	*%t*
Local_Rx_End	
Local_Tx_Begin	
Tx_Antenna_Height:	*(%f [%s][%s])*
Tx_Antenna_Type:	*%s*
Tx_Antenna_Orientation:	*(Direction: %f %f %f) // Omni // (Target: %t)*
Tx_Location:	*%t*
Tx_Power:	*%f [%s]*
Tx_Frequency:	*%f [%s]*
Local_Tx_End	
Support_Type:	*Point // Track // Path // Grid*
Location_Specifier:	*The format depends on the support type.*
Evenly_Spaced_X:	*%f %f %d*
Evenly_Spaced_Y:	*%f %f %d*
Values_Begin	
%f %f %f %f ... %f	
Values_End	
Data_End	
...	
[Data Record #N]	
Data_Begin	
...	

Table 5-1. The SISP General Data Format

SISP General Data Format
...
Data_End

*. The format specifiers used in the table are format specifiers used in ANSI C as described in [Har91].
†. %t is used to denote a terrain specifier whose format depends on the coordinate system used. For instance, when the geographical coordinate system is used, the format specifier represents the format: %d %d %f %c %d %d %f %c; when the UTM system is used, the format specifier represents the format:%f %f %d.

The following sections present an outline of the SISP General Data Format. A detailed discussion of the format is provided in [Sis94].

5.5.1 The Prologue

The prologue contains a mandatory header section and a variable section containing the model specific parameters, receiver information and transmitter information. The header section is invariant and is delimited by the keywords *Header_Begin* and *Header_End*. The receiver definition and the transmitter definition occurring in the prologue are optional and are demarcated by the keywords *Rx_Begin* and *Rx_End* and *Tx_Begin* and *Tx_End* respectively. Yet another optional field in the prologue is the section containing model specific information. The fields in this are dependent on the propagation prediction model associated with the data sets in the file. This section is delimited by the keywords *Model_Specific_Information_Begin* and *Model_Specific_Information_End.*

The significance of the fields in the header section are explained in this paragraph. The *Description*, *Date_Stamp* and *Time_Stamp* fields provide a description of the data, and the date and time attached to the data. The *System* field specifies the wireless system associated with the data set. The *Variable_Station* field determines whether the receiver or the transmitter varies in each data record or, if this varies from one record to another. The *Data_Type* field specifies the quantity that the data set represents. For instance, typical contents of this field could be one of power delay profile, RMS delay spread, local average

received power, among other values. The *Support_Type* field, can have any of these values: *Point*, *Grid*, *Path* or *Track*. The significance of these support types is discussed in Section 5.4. The *Loc_Format* specifies the format of location specifiers. The *Reference_Pt_Loc_Format* describes the coordinate system used for the data. The *Reference_Point* field specifies the coordinates of the reference point to which the locations specified in the file are referenced. The *Building_Database* and *Terrain_Database* fields contain the name of the terrain data file and the building data file associated with the data. The entries for the file names are case sensitive and should conform to SunOS file naming conventions. The *Num_Data_Rec* field contains the number of data records in the file. The *Data_Origin* field specifies whether the data is obtained from measurements, from propagation prediction routines, or from post processing programs. The last field in the header section, the *Units* field contains the units for each dimension of the data. The various fields in the header should always be present and should appear in the same sequence as shown in Table 5-1.

Following the header section in the prologue are the receiver section, the transmitter section and the model specific information section. The receiver and transmitter sections each contain information about the height of the antenna, the type of the antenna, the orientation of the antenna, and the location of the antenna. Additionally, the transmitter section contains the fields which specify the transmitted power and the transmitting frequency of the antenna. The parameters in these sections can occur in any order within the section delimiters.

The section which contains the model specific information is present for data files in which the *Data_Origin* field contains the value *Predicted* or *Processed*. The information contained in this section is, as stated earlier, variable and dependent on the propagation prediction model associated with the data sets in the file. The following section describes the format of the data section.

5.5.2 The Data Section

The data section can contain multiple data records, each of which is delimited by

the keywords *Data_Begin* and *Data_End*. The number of data records in the data section of the file is contained in the header field *Num_Data_Rec*. Each of the data records, can contain optional local transmitter and receiver definitions. However, if the prologue does not contain transmitter and receiver definitions these sections become mandatory in each data record. These sections are delimited by the keywords *Local_Rx_Begin* and *Local_Rx_End*, and *Local_Tx_Begin* and *Local_Tx_End*. They can occur in the data record in any sequence, but must precede the *Support_Type* and *Location_Specifier* fields, and the actual data. A maximum of one local transmitter definition and one local receiver definition may be given in each data record.

The format of the contents of the local transmitter and receiver definitions are similar to those in the receiver and transmitter sections of the prologue. The various fields within these sections can occur in any sequence. The values provided for the parameters of the receiver and the transmitter in the local definitions override the values provided in the prologue. However, these values are applicable only to the data record within which they occur. If the succeeding data record does not contain any local definitions for the transmitter or the receiver, the values provided in the prologue apply to that data record.

The *Support_Type* field is optional and may be present if it is different from the *Support_Type* specified in the prologue. Like all local definitions this overrides the value provided in the prologue. The *Location_Specifier* field contains the coordinates of the location associated with the data record. The format of the location specifier is dependent on the value of the *Support_Type*. The valid formats for the contents of this field are discussed in detail in [Sis94]. There are two other optional fields *Evenly_Spaced_X* and *Evenly_Spaced_Y* which are used to specify the evenly spaced X or Y components in multidimensional data sets. Following these fields in each data record is the data set, enclosed within the delimiters *Values_Begin* and *Values_End*. The formats of the data set depend on the dimensionality of the data and the contents of the optional fields *Evenly_Spaced_X* and *Evenly_Spaced_Y*. The details of the formats of the data set are provided in [Sis94].

This section has presented an overview of the SISP General Data Format. A com-

prehensive description of this data format, and the current revision of this data format can be found in [Sis94]. Data files in the SISP General Data Format are converted to a binary format for the purpose of visualization. The following section describes the Hierarchical Data Format (HDF) which is the binary data format used by SISP.

5.6 Hierarchical Data Format (HDF) - An Introduction

HDF is a binary, public domain, multiobject file format specification for the storage and transfer of images and floating point data between machines [Hdf90]. The design of this format allows self-definition of data content, and easy extensibility for future enhancements or compatibility with other data formats. HDF originated at the National Center for Supercomputing Applications (NCSA), at the University of Illinois at Urbana-Champaign, where it was developed as a solution to the problem of sharing data across different computer platforms.

Storage and retrieval of scientific data is a complex problem. Scientific data is typically generated and processed across different platforms using various software packages. These data files might differ from one another in the assortment of data that they contain, even for the same application. Often the files may be conceptually related but they may be physically separated. HDF's general purpose file structure addresses these issues by providing the following features:

- Enables programs to obtain information about the data from the file itself
- Allows storage of different mixtures of types of data and related information in different files
- Standardizes the formats and descriptions of many types of commonly used data sets
- Accommodates virtually any kind of data by defining new tags or new combinations of tags

HDF files can store floating point data, scaling information, color images, text

annotation, and other items. These files are self-describing in that for each data object in an HDF file, there are predefined tags that identify such information as the type of data, the amount of data, its dimensions, and its location in the file. Thus the structure of these files make its possible to comprehend the contents of the file, from the information stored in the file. Self-description thus enables data of different types to be stored in a single HDF file.

NCSA maintains and distributes a public domain software library to read and write the HDF format. It runs on a variety of computers including the Macintosh, Sun, VAX, Silicon Graphics, etc. The basic code is written in C, with both C and FORTRAN bindings supported for making calls to the HDF libraries. The HDF specifications and the HDF library are in the public domain.

5.7 Basic Structure of HDF Files

This section discusses the organization of HDF files and describes the components that compose these files.

5.7.1 File Header

The first component of an HDF file is the file header which takes up the first four bytes of the file. The file header is a signature that indicates that the file is an HDF file. Specifically it is the 32-bit magic number formed by the four characters ^N, ^C, ^S, ^A. Thus the hexadecimal value of the header is 0E031301.

5.7.2 Data Object

The basic building block in an HDF file is the data object, which contains both data, and information about the data. A data object has two parts: a 12-byte data descriptor (DD) and a data element. Figure 5-2 shows examples of data objects.

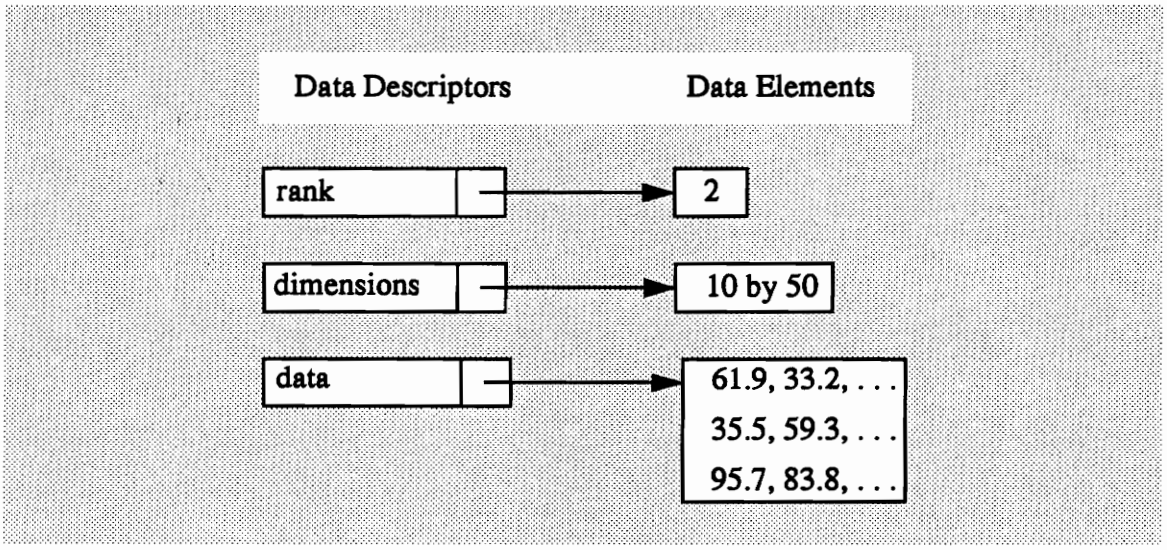

Figure 5-2. Examples of data descriptors and data elements. The rank descriptor denotes the number of dimensions in the data set. The dimensions descriptor specifies the magnitude of each dimension and the data descriptor refers to the actual data.

5.7.2.1 Data Descriptor (DD)

A data descriptor, of length 8 bytes, has four fields: a 16-bit tag, a 16-bit reference number, a 32-bit data offset, and a 32-bit data length. Table 5-2 lists the various fields in the data descriptor with a brief description for each of them.

Table 5-2. Fields in a data descriptor

FIELD	DESCRIPTION
Tag	Designates the type of data in a data object
Reference number	Uniquely distinguishes the data object from others with the same tag.
Data identifier	Tag/ref pair which uniquely identifies a data object in the entire file.
Offset	Offset, in bytes, of the data object from the start of the file.
Length	Length of a data object.

A detailed description of each field in the data descriptor is provided in the following paragraphs.

- **Tag**

A tag is a part of a data descriptor that specifies what kind of data is contained in the corresponding data element. A tag is a 16-bit unsigned integer in the range 0-65535. However, these tags are assigned specific names, defined in header files, which are used to refer to them. If a data descriptor has no data element then the value of its tag is NULL. A tag may never be zero.

- **Reference Number**

Reference numbers are 16-bit unsigned integers. These reference numbers are used to identify the data object in the file. For each occurrence of a tag in an HDF file, a unique reference number is stored with the tag in the data descriptor.

- **Data Identifier**

The combination of a tag and the corresponding reference number uniquely identifies the data object in the file. The tag/reference pair is referred to as a data identifier.

- **Data Offset**

The data offset represents the offset in bytes, of the corresponding data element, from the start of the file. The offset is a 32-bit unsigned integer.

- **Data Length**

The length field of the data descriptor is a 32-bit unsigned integer. It gives the size of the data element in bytes.

Figure 5-3 depicts the different fields that compose a data descriptor.

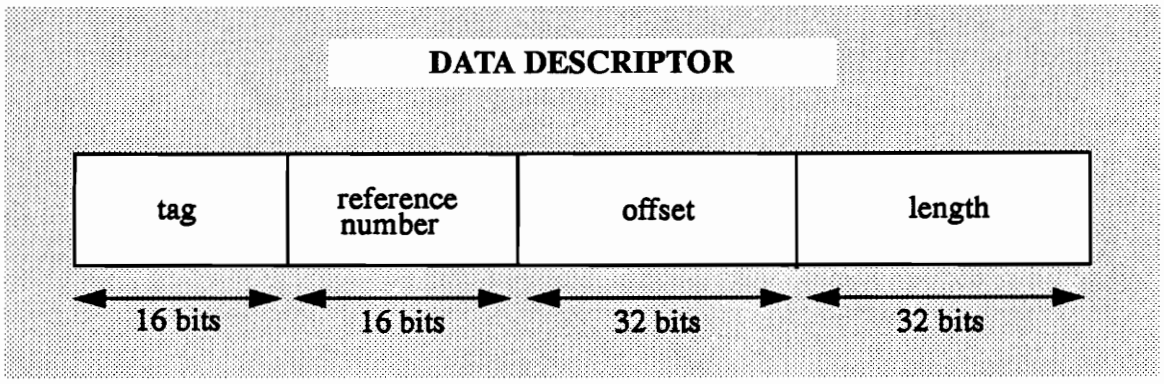

Figure 5-3. Fields composing a data descriptor.

5.7.2.2 Data Descriptor Blocks

Data descriptors are stored physically in a linked list of blocks called data descriptor blocks (DD blocks). The block also contains a data descriptor header at the start of the block. The data descriptor header has two fields-a block size field and a next block field. The block size field is a 16-bit unsigned integer that indicates the number of data descriptors in the following data descriptor block.

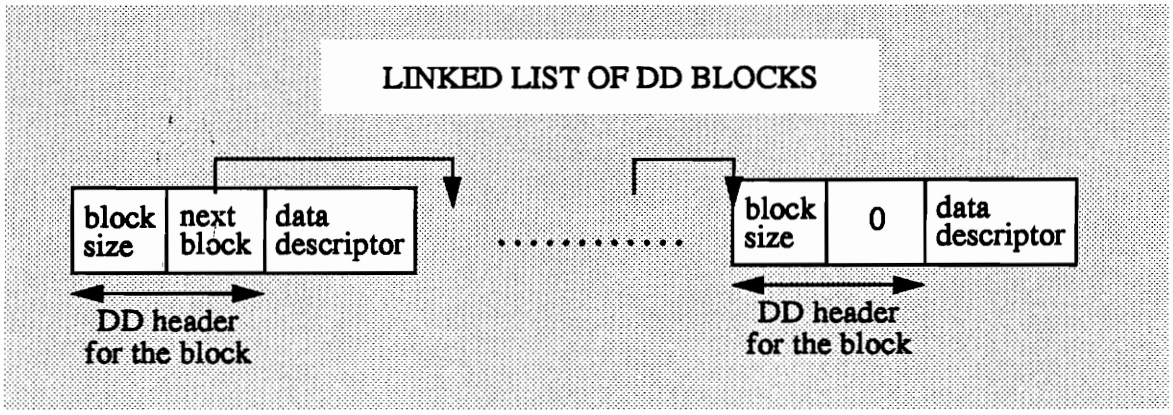

Figure 5-4. Sample data descriptor blocks. The blocks comprise of a header and data descriptors. The block size field in the header indicates the size of the DD block and the next block field contains the offset of the next DD block. The last block in the list contains a 0 in the next block field.

The next block field is a 32-bit unsigned integer specifying the offset of the next data descriptor block, if there is one. The individual components of a data descriptor block are depicted in Figure 5-4. All of the data descriptors in a data descriptor block are assumed to contain significant data unless they have a tag NULL.The last data descriptor header in the list contains a 0 in its next block field.

5.7.2.3 Data Element

A data element is the raw data part of a data object. Its basic type is determined by the tag, but other interpretive information, such as the rank, dimension and data type, may be required before the data can be correctly interpreted. Each data element is stored as a set of contiguous bytes starting at the offset, given in the corresponding data descriptor.

5.7.3 Sets

A set is a group of tags in an HDF file, designed to serve a particular user requirement. Each HDF set is defined in terms of a minimum collection of data objects that must be present when the set is used. One such set, for instance, is the scientific data set. The data set contains several tags which are used for storing information about scientific data.

For instance, every scientific data set must contain at least the following data objects: a data dimension record for the data, which includes the rank (number of dimensions) and the size of each dimension; a data record, which contains the raw data; a scientific data group record, which identifies all the scientific data set objects.

In addition, optional data objects can be present in the set. For instance, a scientific data set may contain the following optional data objects: a labels record, which contains labels for each of the dimensions; a units record, which contains the units for each of the dimensions and the data; a range record, which contains the maximum and minimum values of the data.

5.7.4 Groups

A group is a data object that explicitly defines all the data objects in a set. As explained earlier, an HDF set is a collection of HDF data objects in a file. The group

serves as a mechanism to explicitly identify those objects that belong to a set.

Since the group is a type of data object, its structure is like that of any other data object. A group data identifier (a tag/reference pair) points to a data element that consists of a collection of data identifiers that make up the corresponding set. A group tag can be defined for any set of data such as raster image sets, palette data sets, or scientific data sets. For instance, scientific data group (SDG) is the group tag used to group members of scientific data sets. SDG data consists of a list of all data identifiers that belong to a particular scientific data set. Thus groups provide a convenient mechanism to locate all the information needed about a particular set.

Organizing the data objects in an HDF file into sets or groups does not alter the fundamental role of the objects. They are still accessible as individual data objects, despite the fact that they belong to a particular set. The same data objects can be grouped differently to provide an alternate view of the data in the file.

The previous sections discussed the basic structure of HDF files and the various components that compose the file. The storage and retrieval of data in an HDF file can be accomplished through the use of interfaces that accompany the standard software distribution. The following sections describe the HDF interfaces available, and which are utilized by SISP.

5.8 Overview of the HDF interfaces.

A calling interface is a library of routines that can be called from an application program in order to store and retrieve information in HDF files. The objective of the interface layers is to minimize the amount of knowledge an HDF user needs to have about the physical file structure. The HDF library consists of three interface layers built upon a physical file format as shown in Figure 5-5.

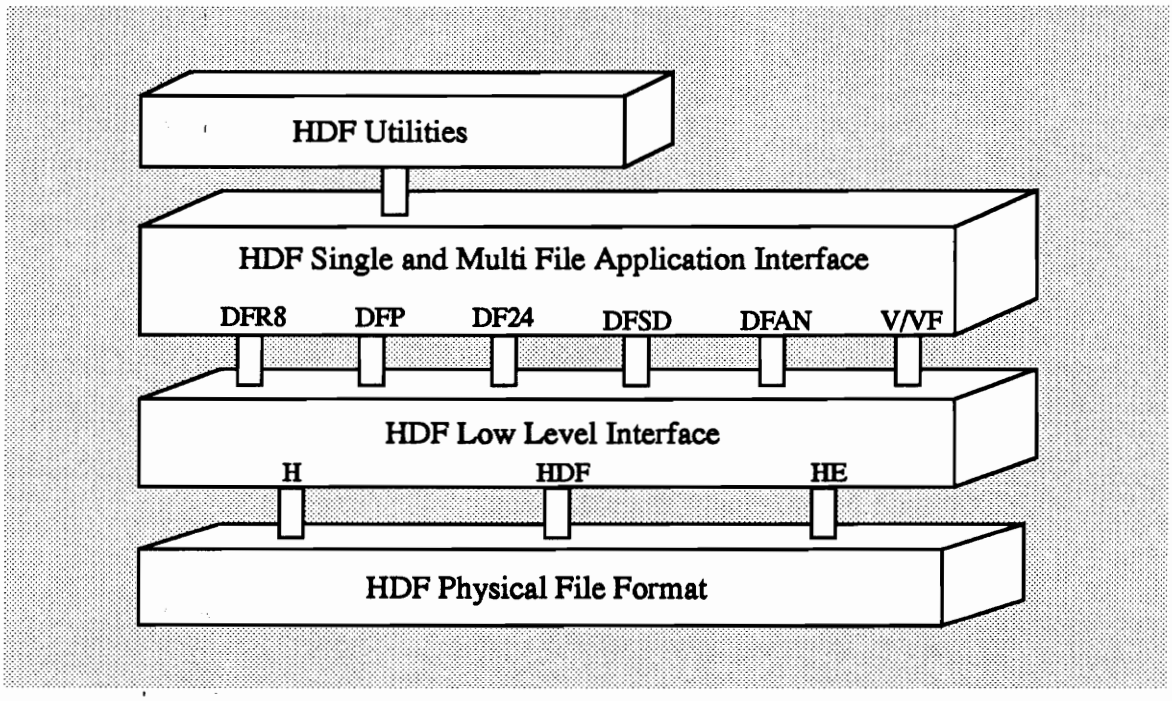

Figure 5-5. HDF interfaces and the organization into layers.

The first layer, or the low level interface, provides support for low level details such as file I/O, error handling, and memory management through its H, HDF and HE routines. The second layer, containing single and multifile application interfaces, consists of a set of independent modules designed to simplify the process of storing and accessing data. The single file interfaces are limited to operating on one file at a time. These interfaces support 8-bit raster image sets (DFR8), 24-bit raster image sets (DF24), palette data (DFP), scientific data (DFSD), and annotation data (DFAN). The multifile interface includes several modules that allow operation on more than one file and data object at the same time. The SD interface is the multiple file version of the single file, scientific data interface (DFSD). The V/VF interface provides a collection of routines for reading and writing customized data sets. The single and multifile interfaces are built on the low level H routines. Finally the highest interface layer includes a collection of command line utilities that operate on HDF files or the data objects that are contained in the file.

The highest level of the interface layer consists of a set of utilities to manipulate HDF files, which are invoked from the command line. Thus, these utilities cannot be integrated with other software packages. The low level interface is intended for programmers who develop HDF interfaces. It is not ideally suited for application programmers because of the detailed knowledge of HDF specifications that it requires. Thus, the second layer of the interface provides the right level of abstraction for users who need to store and retrieve data in HDF. The scientific data set routines (DFSD) and the annotation routines (DFAN) have been utilized to import measured and predicted data into the SISP database. The following sections provide a description of these interfaces.

5.8.1 The Single File Scientific Data Interface (DFSD)

The single file scientific data interface provides a collection of routines for reading and writing data arrays of arbitrary rank and number type. Any array accompanied by a record of its rank and number type qualifies as a scientific data set. Scientific data sets may also include predefined attribute records. Attribute records describe one particular facet of the environment from which the scientific data or its dimensions were obtained. Every function in this single file interface begins with the prefix DFSD. The DFSD interface comprises of function calls, all of which are listed in Table 5-3. The DFSD routines are divided into five categories:

- **Read and write routines** create and read scientific data sets by defining their rank (dimensionality of data set), dimensions (magnitude of each dimension), number type (data type of values in data set), calibration information (such as calibration errors associated with measurements), and data range (maximum and minimum values of data).
- **Slab routines** read and write sub-arrays of scientific data to data sets that already exist in a file.
- **Data set attribute routines** are used to describe the format (format to be used when displaying the data), units (units for the data values), and labels (name of the quantity represented by the data) assigned to a data set. All attributes are optional.

- **Dimension attribute routines** are used to read and write the format, units, and labels, assigned to the dimensions in a data set.
- **General access routines** are used to determine the reference number of a data set, return the number of data sets in a file, and other similar operations.

Table 5-3. Functions available in the DFSD interface

Purpose	Function Name	Description
Reading and writing scientific data sets	DFSDadddata	Appends a data set to the file
	DFSDgetcal	Retrieves calibration information for the data set
	DFSDgetdata	Retrieves the next data set in the file
	DFSDgetdims	Returns the number and dimensions of the next data set
	DFSDgetNT	Determines the number type for the data in the data set
	DFSDputdata	Overwrites the data in a file with new data
	DFSDsetcal	Sets the calibration information for the data set
	DFSDsetdims	Sets the rank and dimensions for the next data set
	DFSDsetNT	Sets the number type for the data set
Reading and writing slabs of scientific data	DFSDendslab	Terminates a read or write slab operation
	DFSDgetfillvalue	Gets the fill value used to complete the data set
	DFSDreadslab	Reads a sub-array of data from a data set
	DFSDsetfillvalue	Sets the fill value to use when completing the data set
	DFSDstartslab	Begins a read or write slab operation
	DFSDwriteslab	Writes a sub-array of data to the data set
Working with data set attributes	DFSDgetdatalen	Gets the length of the attributes assigned to the data
	DFSDgetdatastrs	Returns label, unit and format data
	DFSDgetrange	Gets the range of data values in the data set
	DFSDsetdatastrs	Sets label, unit and format data
	DFSDsetlengths	Sets the length for the data set and dimension attributes
	DFSDsetrange	Sets the range of values for the data set

Table 5-3. Functions available in the DFSD interface

Purpose	Function Name	Description
Working with dimension attributes	DFSDgetdimlen	Length of attributes assigned to the dimensions
	DFSDgetdimscale	Returns the scale for a dimension
	DFSDgetdimstrs	Returns label, unit and format for a dimension
	DFSDsetdimscale	Sets the scale for a dimension
	DFSDsetdimstrs	Sets label, unit and format for the dimension
General access	DFSDclear	Clears all possible set values
	DFSDlastref	Returns the ref number of last data set accessed
	DFSDndatasets	Returns the number of data sets in the file
	DFSDpre32sdg	Checks if the data set was created before HDFv3.2
	DFSDreadref	Locates a data set with a specified ref number
	DFSDrestart	Sets access to the first data set in the file
	DFSDlastref	Returns the ref number of the last data set accessed
	DFSDwriteref	Assigns a specific ref number to the next data set

5.8.2 The File Annotations Interface (DFAN)

The annotation interface provides a collection of routines for reading and writing test strings assigned to HDF data objects or files. Annotations consist of labels and descriptions. A label is any sequence of characters terminated by a null character (character from the ASCII set with a value of 0). A description is any sequence of characters that can contain one or more null characters. Every function in the annotation interface begins with the prefix DFAN. The DFAN routines are listed with brief descriptions in Table 5-4. The DFAN routines can be divided into the three following categories:

- **Write routines** are used to add file labels, file descriptions, object labels and object descriptions to a file.
- **Read routines** are used to determine the length of an annotation, and to read annotations from a file.
- **General access routines** are used to list the labels for a given tag, and to return the

reference number of the last annotation accessed.

Table 5-4. Functions available in the DFAN interface

Purpose	Function Name	Description
Writing annotations	DFANputlabel	Stores an object label
	DFANputdesc	Stores an object description
	DFANaddfid	Adds a file label to the file
	DFANaddfds	Adds a file description to the file
Reading annotations	DFANgetlablen	Returns length of an object label
	DFANgetlabel	Returns label object
	DFANgetdesclen	Returns length of object description
	DFANgetdesc	Returns an object description
	DFANgetfidlen	Returns length of file label
	DFANgetfid	Returns file label
	DFANgetfdslen	Returns length of file description
	DFANgetfds	Returns file description
General access	DFANlablist	Returns a list of labels for a tag type
	DFANlastref	Returns ref of last annotation accessed

5.8.3 The Multifile Scientific Data Set Interface (SD)

The HDF multifile interfaces consist of several modules that permit operations on more than one file and data object simultaneously. The SD interface is the multiple file version of the scientific data set interface (DFSD). It also includes expanded support for interoperability with the netCDF library. Like the single file interfaces, the multifile interfaces are also built on low-level H routines. However, unlike single file operations, the operations that are performed via a multifile interface are not implicitly preceded by a *Hopen()* and followed by *Hclose()*. Instead, each series of operations on a file must be preceded by an explicit call to open and close the file. Once the file is opened, it remains open until an explicit call is made to close it.

The scientific data set interface provides a collection of routines for reading and writing arrays of arbitrary dimensions and data types. Multidimensional arrays accompanied by a record of their dimension and number type are tagged as scientific data sets. Under the multifile interface, the data sets may include optional, predefined or user defined attribute records. Every function in the multifile scientific data set interface begins with the prefix SD. The functions are listed in Table 5-3 with brief descriptions. The SD routines are divided into the following four categories:

- **Access routines** initialize and close the SD interface
- **Create, read and write routines** create, read, and write data sets by defining or reading their dimensions, rank, data type etc.
- **Dimension attribute routines** describe the attributes of data set dimensions.
- **General attribute routines** manage information about local attributes (relevant to a single data set) and global attributes (attributes assigned to a file). The interface supports user defined local and global attributes.

Table 5-5. Functions available in the SDinterface

Purpose	Function Name	Description
Access SD interface	SDend	Closes the file and cleans up memory
	SDstart	Initializes the SD interface

Table 5-5. Functions available in the SDinterface

Purpose	Function Name	Description
Reading and writing data sets	SDcreate	Create a new data set
	SDendaccess	Dispose of a data set ID
	SDfileinfo	Return info about the file contents
	SDgetinfo	Return information about a data set
	SDidtoref	Returns a reference number for a named data set
	SDiscoordvar	Distinguishes between a data set and coordinate variable
	SDnametoindex	Returns an index for a named data set
	SDreaddata	Reads a slab of information from a data set
	SDselect	Gets the sd_id for a specified data set
	SDwritedata	Write a slab of data for a data set
Working with dimensions	SDdiminfo	Gets the information about a dimension
	SDgetdimid	Gets a pointer to a dimension
	SDgetdimscale	Retrieves scale values for a dimension
	SDsetdimname	Associate a name with a dimension
	SDsetdimscale	Store the values of this dimension

Table 5-5. Functions available in the SDinterface

Purpose	Function Name	Description
Working with attributes	SDattrinfo	Get information about an attribute
	SDfindattr	Find the index for a named attribute
	SDgetcal	Get the calibration information
	SDgetdatastrs	Return the data strings for a data set
	SDgetdimstrs	Get the attribute strings for a dimension
	SDgetfillvalue	Read the fill value if it exists
	SDgetrange	Get the range of values for the data set
	SDreadattr	Read the values of an attribute
	SDsetattr	Define a new attribute
	SDsetcal	Set the calibration information
	SDsetdatastrs	Set the attribute strings for a data set
	SDsetdimstrs	Set the attribute strings for a dimension
	SDsetexternalfile	Store data in an external file
	SDsetfillvalue	Set the fill value for this data set
	SDsetrange	Set the max. min. values for the data set

5.9 Conversion from GDF To HDF

5.9.1 Version 1.0

The conversion of data from GDF to HDF is primarily to aid the development of an efficient access mechanism for programs that display the data. The first version of the conversion program was developed to evaluate the suitability of the format for storing measured and predicted data and to ascertain the capability of the programming interfaces. This process provided an understanding of the structure of the HDF files and the programming paradigm conformed to by the interface routines. Changes to the structure of the GDF, made since the first version, have necessitated the development of a new version of the conversion programs. The following paragraphs provide a brief description of the first version of the conversion program.

The primary objective of the conversion program is to ensure that there is no loss of information in the conversion process from GDF to HDF. Although all the information in the GDF file is not used for display purposes, the information is retained in the conversion process to allow for back conversion from HDF to GDF. The contents of the GDF file can be grouped as belonging to the header section, the variable parameters section or the data records section. Figure 5-6 illustrates the mapping from GDF file contents to the HDF data objects and the specific functions that accomplish the conversion. The header section of the GDF file is stored as the description of the HDF file, the variable parameters section is stored as a description of the data set, the data record is stored as the data object, and the location specifier for the data record is used to label the HDF data object.

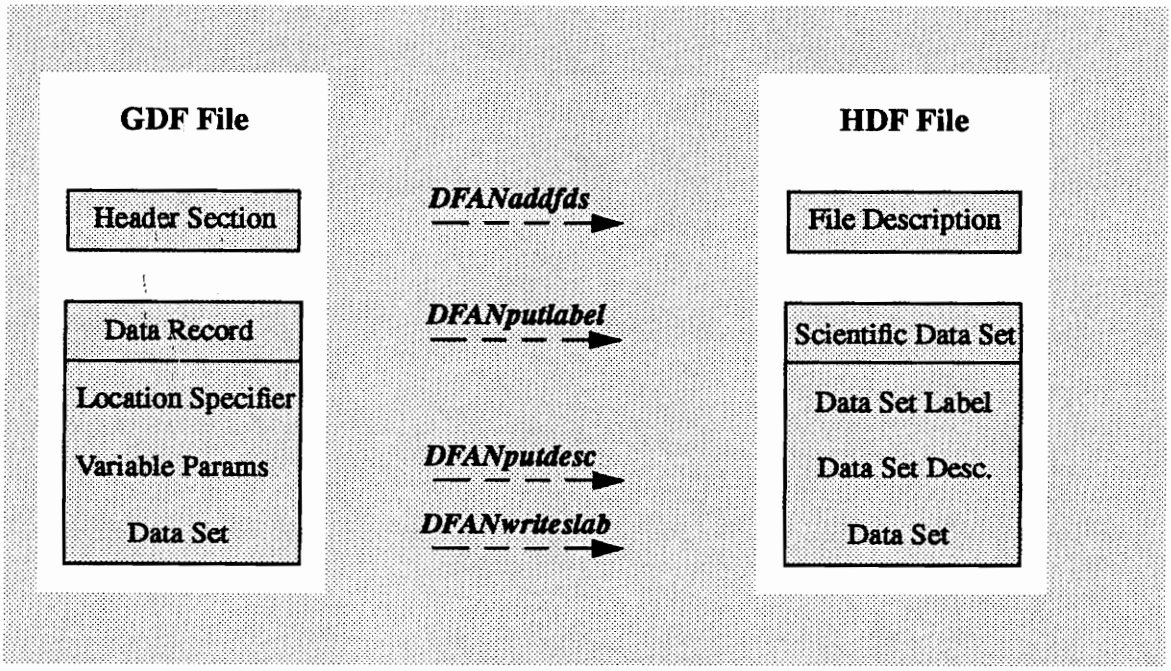

Figure 5-6. Mapping GDF file contents to HDF data objects - version 1.0. This version of the conversion program uses the DFSD and DFAN interfaces.

The conversion process can be represented by the flowgraph as shown in Figure 5-7. The figure shows only the HDF function calls that are invoked for the conversion. The HDF function *DFANaddfds()* is invoked to store the header section of the GDF file as the

description of the HDF file. After the header section has been stored, the data records are read iteratively from the GDF file and stored in the HDF file. The following paragraph describes this process.

Since multiple data objects can be stored in the HDF file, a label is used to identify the data objects in the file. The label is composed of the easting, northing and UTM zone values occurring in the location specifier of each data record. The variable parameters specific to each data record are concatenated to form the description of the data record. The function *DFSDclear()* is invoked at the start to initialize the relevant data structures for each record. The functions *DFSDsetdims()* and *DFSDsetNT()* are invoked next, to set the rank, dimension, and data type for the data set. After specifying the dimensions of the data set, the slab write operations, supported by the DFSD interface, are invoked to create the data set. The slab read and write operations provide the flexibility to read and write segments of the data set. The function *DFSDwriteslab()* is invoked to store the data values composing each dimension of a multidimensional record. The DFSD interface specifies that the functions *DFSDstartslab()* and *DFSDendslab()* are invoked to mark the start and end of the slab write operations, to ensure precise storage of data. The reference number of the data set written is determined and used as an argument for the DFAN functions, *DFANputlabel()* and *DFANputdesc()*. These functions are called to associate a label and a description with the data record. The conversion process is terminated once all the data records in the HDF file have been processed.

This paragraph discusses the retrieval of the header section and a data object from the HDF file. This process is illustrated in the flowgraph in Figure 5-7. The location specifier which identifies the data set is supplied. The header section of the GDF file, stored as the description of the HDF file, can be retrieved by invoking the function *DFANgetfds()*. After retrieving the header section, the program determines the number of scientific data sets stored in the file, by invoking the function *DFSDndatasets()*. It reads the labels for all the data sets in the file by calling the function *DFANlablist()*.

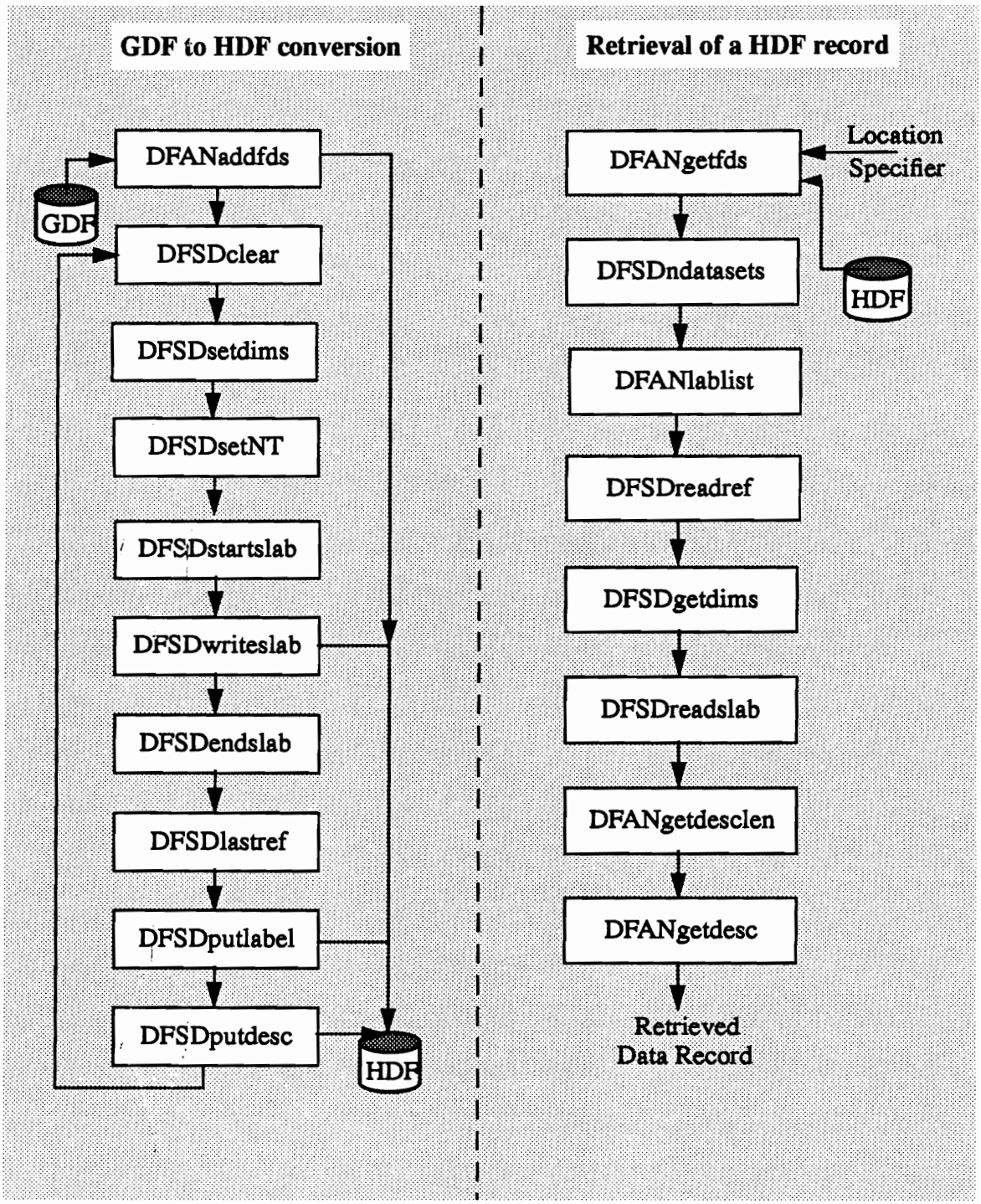

Figure 5-7. Flowgraph of HDF function calls - version 1.0 . Illustrates the process of conversion from GDF to HDF and the retrieval of a data record from the HDF file.

The label, used to identify the data set, is composed using the easting, northing and UTM zone of the location specifier. The reference number associated with the label is determined and the function *DFSDreadref()* is invoked to fix this as the reference number of the data set to be read. The rank and dimensions of this data set are determined using the function *DFSDgetdims()*. The data values, along each dimension, are read using the function *DFSDreadslab()*. The description for each data set contains information about the variable parameters associated with each record. This is read by invoking the function calls *DFSDgetdesclen()* and *DFSDgetdesc()*. Thus the header section and the entire data set can be retrieved from the HDF file. If this process is performed for all the objects in the data set, the back conversion from HDF to GDF can be achieved.

Although a new version of the conversion programs have been developed, this version of programs can be used for accessing HDF files when multiple files need not be opened simultaneously.

5.10 Version 2.0

This section discusses the current version of the program developed for data conversion from GDF to HDF. The reasons leading to the development of a new version of the conversion program are listed below.

- The GDF has undergone fundamental structural changes since the first version.
- GDF can accommodate local definition sections, for transmitter and receiver parameters, in the data record. These optional sections are present within the data record when a parameter has to be redefined, or has been omitted from the header section, or is specified as variable in the header section. Access to the information in these sections would be inefficient if they were concatenated and stored as description strings.
- Changes in the SISP framework have made the use of multiple windows, to display the measured and predicted data, possible. This would require multiple data

files to be open simultaneously. However, the DFSD interface cannot support multiple open files.

The current version of the conversion program uses the SD multifile interface. The SD interface and the complete documentation has become available in the current HDF version-3.3r3. The SD interface is functionally different from the DFSD and DFAN interfaces, although they have been developed from the same HDF library. The difference is primarily in its capability to support user defined, global and local attributes, in addition to its support for multiple open files.

An attribute consists of a value or a group of values, of several data types, identified by a name. The scope of these attributes can be global or local depending on whether they have been assigned to the file or to a specific data set. The interface supports several predefined attributes and can also accommodate user defined attributes. The global attributes for a file can be created by calling the function *SDsetattr()* with the file identifier as its first argument. The local attributes can be defined by invoking the same function but with the identifier for the data set as the first argument.

Figure 5-8 shows the mapping between the contents of the GDF file and the HDF data objects and the SD routines that accomplish the conversion. The header section of the file is stored as a set of global attributes, the variable parameters sections are stored as local attributes assigned to the data set, and the data in each record is stored as a data set in the HDF file. The data sets are identified by a name which is composed of the easting, northing, and UTM zone values present in the location specifier of the data record.

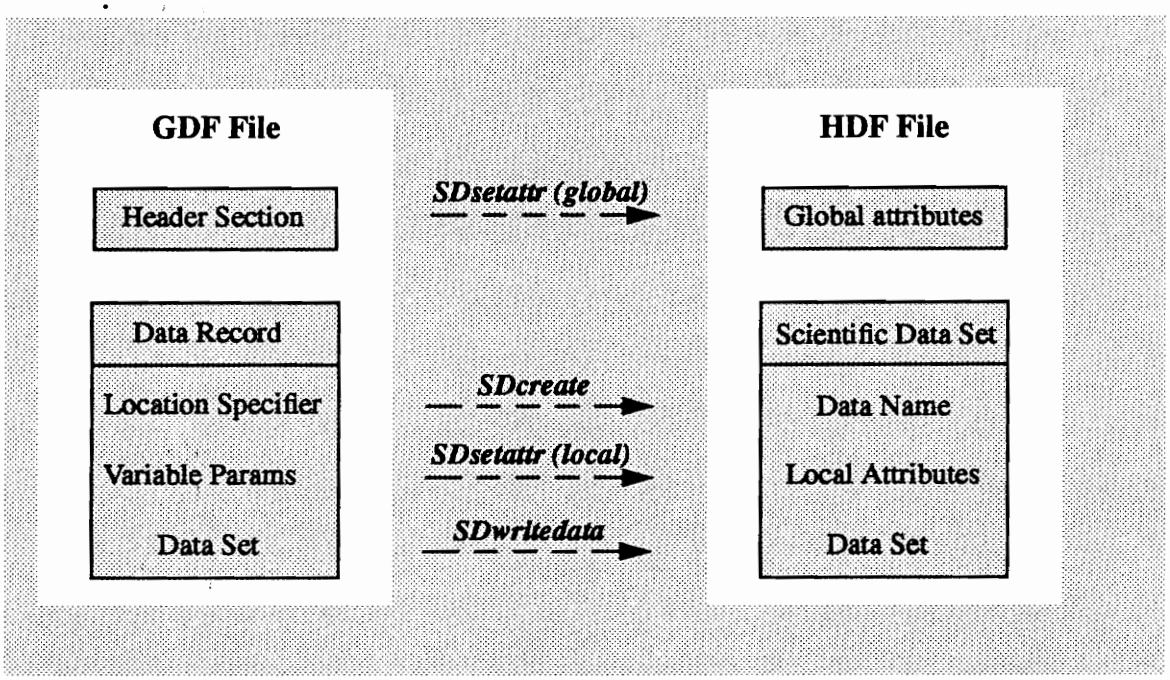

Figure 5-8. Mapping contents of GDF files to HDF files - version 2.0. This version utilizes the multifile SD interface

The conversion process can be represented by the flowgraph shown in Figure 5-9. The figure shows only the SD function calls which are utilized to perform this conversion. The function *SDstart()* is invoked at the start of any HDF operation via the SD interface. The name of the HDF file and the mode in which it is to be opened are passed as arguments to this function. This routine initializes the SD interface and returns an *sd_id* for the file. The header section of the GDF file is read and is completely stored in the HDF file by iterative calls to the function *SDsetattr()*. This function defines global attributes to store the information in the HDF file.

The data records in the GDF file are stored as data objects in the HDF file. A data object in the HDF file is created by a call to *SDcreate()*. The name of the data object to be created and the file identifier of the file, in which it is to be stored, are passed as arguments to this routine. The name of the data set to be created is composed from the easting, northing, and UTM zone values present in the location specifier of the data record.

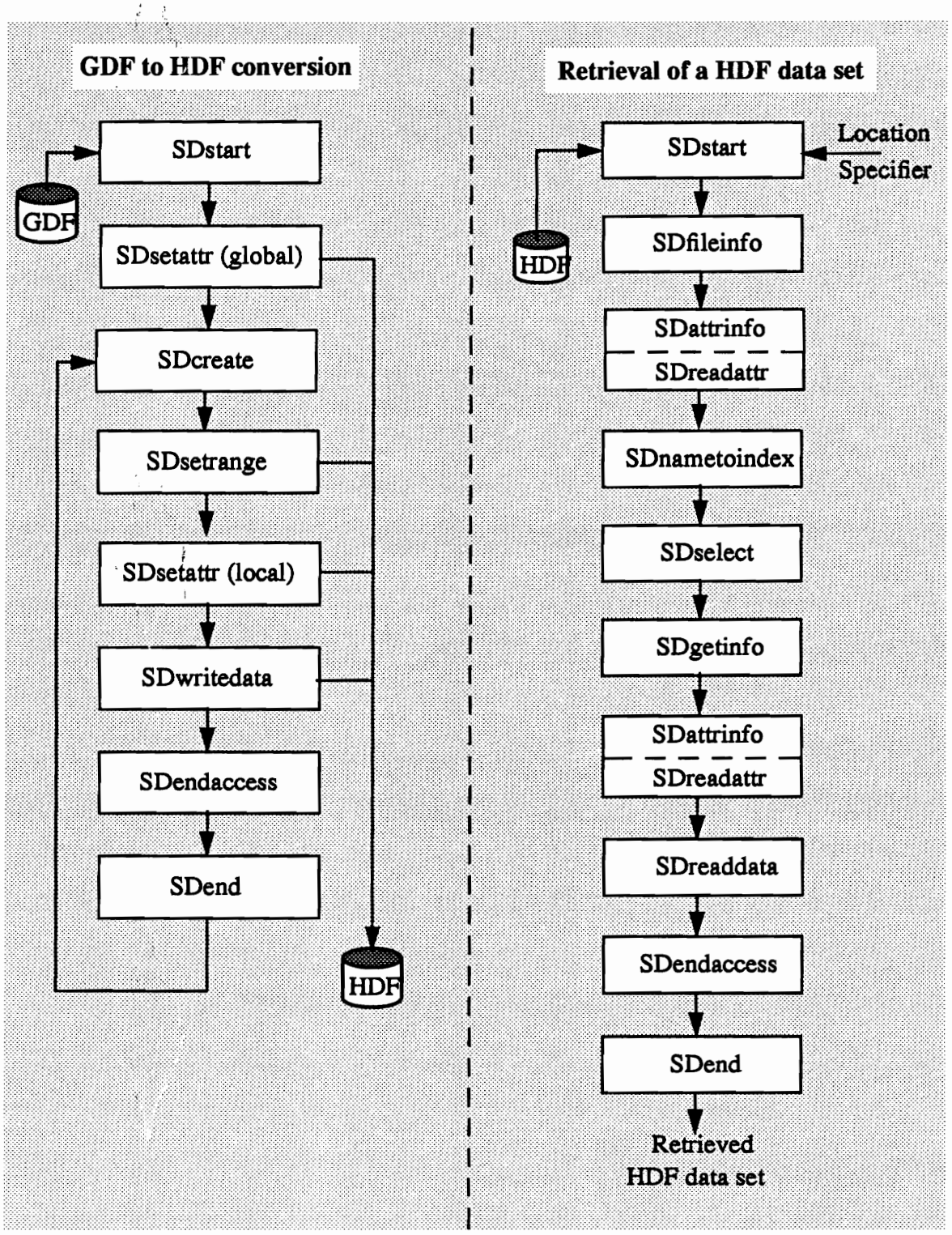

Figure 5-9. Flowgraph of HDF function calls - version 2.0.

The other arguments to this routine are the rank, the dimension and the data type of the values in this data set.This function returns a *sds_id* which is the identifier for the particular data object created. The local definition sections in the data records are stored as local attributes in the HDF file. The local attributes are created by calling the function *SDsetattr()* with the *sds_id* as the first argument to the function.

The dimensionality of the data set and the magnitude of each dimension is obtained from the parsing routines. The function *SDwritedata()* is invoked to write the values which compose one dimension of the data set. The range of data in the data set is stored in the HDF file with a call to the function *SDsetrange()*. When the data has been written to the HDF file the function *SDendaccess()* is called to indicate the end of the operation for the data set. The same process is used for each of the data records in the GDF. If all the records have been converted, the function *SDend()* is called to free the file identifier *sd_id* and free memory associated with the open file.

This paragraph explains the process of reading the header section and a data set from the HDF file. The HDF file is opened and the SD interface is initialized with the *SDstart()* function call. The function *SDfileinfo()* is invoked to determine the number of data sets and global attributes in the HDF file. The global attributes are indexed by a continuous sequence of index numbers starting from zero. Thus the header section of the file is recovered by iterative calls to the functions *SDattrinfo()* and *SDreadattr()*. A location specifier is supplied to access the data set associated with the location. The easting, northing, and UTM zone values present in the location specifier is used to compose the name of the data set. The function *SDnametoindex()* is called to obtain the index for the data set. This index is passed as an argument to the function *SDselect()* which returns an identifier (*sds_id*) for the data set. The function *SDgetinfo()* is invoked to obtain information about the data set. The rank, dimension, and the number of attributes for the data are returned by this function. The rank and dimension of the data set are used to evaluate memory requirements for the data set. The data set is read from the file by calling the function *SDreaddata()* to obtain the values. The data values are read until the end of the data record is

encountered. The local attributes for the data are the local redefinition sections occurring within each data record. This is read from the HDF file by invoking the functions *SDattrinfo()* and *SDreadattr()*. This paragraph described the procedure to read one data set from the HDF file. The conversion from HDF to GDF can be performed by applying the procedure for all the data sets in the HDF file.

This chapter presented a discussion of the data formats used to import and store measured data for SISP. It introduced several formats currently in use for the storage of scientific data. An overview of the SISP General Data Format (GDF) was provided. The Hierarchical Data Format (HDF) was presented and a detailed discussion of the programming interfaces for this format was provided. The conversion programs to convert from GDF to HDF (both versions 1.0 and 2.0) were described extensively in this chapter.

Chapter 6. Conclusions and Future Work

6.1 Summary of Work

This work has presented the software developed to establish a framework for site specific propagation prediction. Chapter 1 provided an introduction to wireless communications systems in general and discussed the significance of propagation prediction. An overview of the resources utilized by the software implementations of the propagation prediction models, and the functional specifications for SISP were presented in Chapter 2.

Chapter 3 described the development of the raster programming library, which has been one of the contributions of this thesis. The raster programming library comprises reusable modules that provides the functionality to handle raster data such as terrain elevation data. Since site specific propagation models manipulate diverse and large databases of geo-referenced data, this chapter provided an introduction to geographical information systems. The salient features of GRASS (Geographical Resources Analysis & Support System) including its process model were discussed. The changes made to GRASS programs incorporated in the raster programming library was also discussed. The capabilities of the programming library that has been developed, and the implementation details of some of the important functions in the library has also been discussed.

Another contribution of this thesis has been the vector programming library that has been developed. Chapter 4 discussed the functionality of this programming library. This library is essentially used for handling and displaying site specific vector data such as building contour data. The program modules that have been developed for the visualization of vector data were discussed in detail. This chapter also presented the difficulties in handling the diverse formats, that building contour and elevation data currently exist, and the standard format that has been defined to overcome this difficulty. A detailed description of the format and the filters that have been implemented to import building data files into the SISP database are also included in this chapter. This chapter also discussed the capabilities of the plotting library that has been developed for displaying vector, RF mea-

surement and propagation prediction data.

The program modules developed to store and retrieve measured and predicted data in the Hierarchical Data Format (HDF), discussed in Chapter 5, has been yet another contribution of this thesis. This chapter commenced with a discussion of formats that are widely in use for the storage of scientific data. An important issue in handling RF measurement data is its availability in multitudinous formats, which, depending on the type of system that has been used to make the measurements, may vary extensively. The importance of a standard format for this data was underscored; the SISP General Data Format (GDF), developed at the Mobile and Portable Radio Group (MPRG), which serves as a standard, ASCII format for importing measured data was discussed. Since, rapid access and retrieval of data stored in the database is critical for display and analysis, the HDF has been adopted for the storage of predicted and measured data in the SISP database. HDF is a binary indexed data format for the storage of scientific data, developed by the National Center for SuperComputing Applications (NCSA). This chapter presented a brief overview of this format and the associated programming interface. It also discussed the functionality of the programs that have been developed to store and retrieve data in the HDF format. The following section suggests some possible extensions to this work.

6.2 Future Work

This work has presented the development of programming libraries for use in SISP. It also demonstrated the use of HDF for storage and retrieval of RF measurement and prediction data. The current programming libraries only support the two-dimensional display of terrain and building data. One possible extension of this work could be to develop the functionality for three-dimensional visualization of terrain and building data. Since the magnitude of the third dimension of terrain and building data is stored in the database, the information necessary for three-dimensional visualization is readily available.

Thus far, the software development has focussed on the development of functions

that aid the execution of outdoor propagation models. Since SISP is intended to serve as a platform for the execution of outdoor and indoor models, an extension of this framework to support the execution of indoor models is essential. The indoor models utilize the structural information within a single building. This information is obtained as AutoCAD DXF files with each file representing the layout of a single floor in a multifloor building. The primary task would be to standardize the format for this data. The development of programs to import this data into the SISP database, and to aid the visualization of multifloor buildings could constitute the other tasks in this extension.

This work has adopted HDF as the binary format for storing measured and predicted data in the SISP database. The programming functions to store predicted and measured data in this format has been developed, and its flexibility to accommodate future changes to the SISP GDF has been illustrated. Several other data formats exist for the storage of scientific data and have been documented in [For92]. These data formats and the programming support that they provide, in the form of function libraries, can be evaluated for use in SISP.

Chapter 7. References

[Acad90] *AutoCAD Release 11*, Reference Manual, Autodesk Inc., 1990.

[Air93] *Phoenix Product Literature,* AirTouch Communications, Walnut Creek, CA., 1993.

[Arc88] *Understanding GIS: the ARC/INFO method*: rev. 6 for workstations, Environmental Systems Research Institute, Longman Scientific and Technical, NY, 1993.

[Ase90] P.J. Asente, et al, *X Window System Toolkit, The Complete Programmer's Guide and Specification*, Digital Press, 1990.

[Bur86] P.A. Burrough, *Principles of Geographical Information Systems for Land Resources Assessment*, Oxford University Press, New York, pp. 41 - 43, 1986.

[Cal88] G. Calhoun, *Digital Cellular Radio*, Artech House, Norwood, MA, 1988.

[Cam84] J Campbell, *Introductory Cartography*, Prentice-Hall, Inc., NJ, 1984.

[Cha93] James J.C. Chang, et al., "Wireless Systems and Technologies: An Overview," *AT&T Technical Journal*, pp. 11-18, Jul.-Aug., 1993.

[Col80] Wilfred P. Cole, Preservation and Planning Series, *"Using the UTM Grid System to Record Historical Sites,"* U.S. Department of the Interior, Heritage Conservation and Recreation Service, 1980.

[Cow88] D.J. Cowen, "GIS versus CAD versus DBMS: What Are the Differences?" *Photogrammetric Engineering and Remote Sensing*, Vol. 54, No. 11, pp. 1551 - 1555, Nov. 1988.

[Cox88] Donald C. Cox, "Wireless Network Access for Personal Communications", *IEEE Communications Magazine*, pp. 96 - 114, Nov. 1988.

[Eas88] J.R. Eastman, "IDRISI: A Geographical Analysis System for Research Applications," *The Operational Geographer*, No. 15, pp. 17 - 21, May 1988.

[Erd91] *ERDAS Software Overview*, ERDAS, Inc., 1991.

[Fol90] J.D. Foley et al., *Computer Graphics: Principles and Practice* (2nd ed.), Addison Wesley, 1990.

[For92] Brand Fortner, *The Data Handbook - A Guide to Understanding the Organiza-*

tion and Visualization of Technical Data, Spyglass Inc., IL, 1992.

[Gla90] Andrew S. Glassner, *Graphics Gems*, Academic Press, Inc., Harcourt Brace Jovanovich, Publishers, NY, 1990.

[Goo91] David J. Goodman, Trends in Cellular and Cordless Communications, *IEEE Communications Magazine*, pp 31-39, June 1991.

[Gre92] L.J. Greenstein, et al., "Microcells in Personal Communications Systems," *IEEE Communications Magazine*, pp.76-88, Dec. 1992.

[Har91] Samuel Harbison and Guy Steel, *C-A Reference Manual*, 3rd ed., Prentice Hall, NJ, 1991.

[Hat80] M. Hata, "Empirical Formula for Propagation Loss in Land Mobile Radio Services," *IEEE Transactions on Vehicular Technology*, Vol VT-28, No. 3, pp.317-325, August 1980.

[Hdf90] *NCSA HDF, Calling Interfaces and Utilities*, Version 3.1, University of Illinois, Urbana Champaign, July 1990.

[Hel87] Gilbert Held, and Thomas R. Marshall, *Data Compression: Techniques and Applications, Hardware and Software Considerations* (2nd ed.), John Wiley & Sons, NY, 1987.

[Hua92] Chia-Chi Huang and Rana Khayata, "Delay Spreads and Channel Dynamics Measurements at ISM Bands," *International Conference on Communications*, Chicago, IL, pp. 1222-1226, June 14, 1992.

[Jon89] Oliver Jones, *Introduction to the X Window system*, Prentice Hall, NJ, 1989.

[Kea89] John S. Keates, *Cartographic Design and Production*, 2nd ed., Longman Scientific &Technical, Copublished with John Wiley & Sons, Inc., NY, 1989.

[Kob93] Bennet Z. Kobb, "Special report - Personal Wireless Communications," *IEEE Spectrum*, pp. 21-25, June 1993.

[Kor92] George Korte, *The GIS book: a practitioner's handbook*, OnWord Press, Santa Fe, NM, 1992.

[Kur92] Thomas Kurner, et al., "Concepts and Results for 3D Digital Terrain-Based Wave Propagation Models: An Overview," *IEEE Journal on Selected Areas in Communications*, Vol. 11, No. 7, pp. 1002-1012, Sep. 1993.

[Kuw85] Moriji Kuwabara, "Present & Future of Mobile Communications", *Japan Telecommunication Review*, Oct. 1985.

[Lar94] David Larsen, "Wireless Technology: How Test Fits In", *Microwave Journal*, pp.63-74, Jan. 1994.

[Lcc93] *Network Engineering Software Overview*, LCC Inc., Arlington, VA., 1993.

[Lee89] W.C.Y. Lee, *Mobile Cellular Telecommunications Systems*, McGraw-Hill, New York, 1989

[Lia85] S.Y. Liao, *Microwave Devices and Circuits*, Prentice-Hall, Englewood Cliffs, NJ, 1985.

[Mac79] H.V. Macdonald, "The Cellular Concept," *Bell System Technical Journal*, Vol. 58, No. 1, pp. 15-41, Jan. 1979.

[Mal92] D.H. Maling, *Coordinate Systems and Map Projections*, 2nd ed., Pergamon Press, NY, 1992.

[Map93] *MapInfo - Desktop Mapping Software*, User's Guide, Mapinfo Corporation, Troy, NY, 1993.

[Mcs93] *MCS Product Literature*, EDX Engineering Inc., Eugene, OR., 1993.

[Mea87] B.K. Meade, "Program for computing Universal Transverse Mercator (UTM) coordinates for latitudes north or south and longitudes east and west," *Surv. Mapping*, Vol. 47, pp. 37-49, 1987.

[Mel93] Lena Melin, et al., "Radio Wave Propagation, A Comparison between 900 and 1800 MHz," *43rd IEEE Vehicular Technology Conference*, Secaucus, NJ, pp. 250-252, May 18, 1993.

[Mog91] P. E. Mogensen, et. al, "Urban Area Radio Propagation Measurements for GSM/DCS 1800 Macro and Micro Cells," *IEEE Conference on Antennas and Propagation*, York, UK, pp. 1-5, April 1991.

[Moh93] W. Mohr, "Wideband Propagation Measurements of Radio Channels in Mountainous Areas in the 1800 MHz Frequency Range," *43rd IEEE Vehicular Technology Conference*, Secaucus, NJ, pp. 49-52, May 18, 1993.

[Mou92] M. Mouly and M-B. Pautet, *The GSM System for Mobile Communications*, Published by the authors, Palaiseau, France, 1992.

[Nov93] *MSRFM Product Literature,* NovAtel Communications Ltd., Calgary, Canada, 1993.

[Nye88] Adrian Nye, *Xlib Programming Manual*, Vol. 1, O'Reilly and Associates Inc., MA, 1988.

[Nye88a] Adrian Nye, *Xlib Reference Manual*, Vol. 2, O'Reilly and Associates Inc., MA, 1988.

[Och89] H. Ochsner, "DECT - Digital European Cordless Telecommunications", *39th IEEE Vehicular Technology Conference*, San Francisco, pp718 - 721, May 1989

[Oku68] Y. Okumura, et al, "Field Strength and its Variability in VHF and UHF Land Mobile Service," *Rev. Elec. Comm. Lab.*, Vol. 16, pp. 825-873, Sept.-Oct. 1968.

[Ore88] Tim O'Reilly, *X Toolkit Reference Manual,* Volume 5, O'Reilly and Associates Inc., MA, 1988.

[Par89] J.D. Parsons, and J.G. Gardiner, *Mobile Communication Systems*, Blackie, London, 1989.

[Pla94] *PlaNet Software Overview*, Mobile Systems International, Chicago, IL., 1993.

[Rai91] Krister Raith and Jan Uddenfeldt, "Capacity of Digital Cellular TDMA Systems," *IEEE Transactions on Vehicular Technology*, Vol. 40, No. 2, pp. 323-332May 1991.

[Raj92] C.S. Rajguru and M.J. Feuerstein, "Application of GRASS in Mobile and Portable Communications," *7th Annual GRASS GIS Users' Conference*, 1992.

[Raj92a] C.S. Rajguru, "Application of GIS in propagation prediction", *Master's Thesis in Electrical Engineering*, Virginia PolyTechnic Institute and State University, Blacksburg, VA, 1992.

[Rap90] T.S. Rappaport, et. al, "900 MHz Multipath Propagation Prediction Measurements for U.S. Digital Cellular Radiotelephone," *IEEE Transactions on Vehicular Technology*, Vol. 39, pp. 132-139, May 1990.

[Rap91] T.S. Rappaport, "Wireless Personal Communications: Trends and Challenges," *IEEE Ant. & Prop. Magazine*, Vol. 33, No. 5, pp. 19-29, Oct. 1991.

[Rap91a] T.S. Rappaport, "The Wireless Revolution", *IEEE Communications Magazine*, Vol. 29, No. 11, pp.52-71, Nov. 1991.

[Rap93] T.S. Rappaport et al., "Radio Propagation Prediction Techniques and Computer-aided Channel Modeling for Embedded Wireless Microsystems," *MPRG Technical Report MPRG-TR-93-02*, Dept. of Electrical Engineering, Virginia Polytechnic Institute and State University, Jan. 10, 1993.

[Rea48] J.C.B. Redfearn, "Transverse Mercator Formulae," *Emp. Surv. Review*, Vol. 9, No. 69, pp. 318-322, 1948.

[Rob78] Arthur Robinson, et al., *Elements of Cartography*, 4th ed., John Wiley & Sons, Inc., NY, 1978.

[Roc85] Marc J. Rochkind, *Advanced UNIX programming*, Englewood Cliffs, NJ, 1985.

[Ros93] J-P Rossi and A.J. Levy, "Propagation Analysis in Cellular Environment with the Help of Models Using Ray Theory and GTD," *43rd IEEE Vehicular Technology Conference*, Secaucus, NJ, pp. 253-256, May 18, 1993.

[Rus91] T.A. Russell, et al, "Use of a Building Database in Prediction of Three Dimensional Diffraction," *42nd IEEE Transactions on Vehicular Technology*, Vol. 40, pp.203-210, Feb 1991.

[Rut72] S.Ruth, and P.Kreutzer, *Data Compression for Large Business Files, Datamation*, Vol. 18, No. 9, pp. 62-66, 1972.

[Sch91] K.R. Schaubach, et al., "Propagation Models for Microcellular Environments Using a Building Database," poster presented at the *First Virginia Tech. Symposium on Wireless Personal Communications*, Blacksburg, VA, June 5, 1991.

[Sch94] K. R. Schaubach and N.J. Davis, "MicroCellular Radio Channel Propagation Prediction," *IEEE Antennas and Propagation Magazine*, Vol. 36, No. 4, Aug. 1994.

[Sis94] *Site Specific Propagation Prediction (SISP) Overview, Standards, and Definitions, Internal Standards Document*, Mobile and Portable Radio Research Group, Virginia Tech., Blacksburg, VA, 1994.

[Sta90] Jeffrey Star, and John Estes, *Geographic Information Systems: An Introduction*, Prentice Hall, NJ, 1990.

[Swa90] R. S. Swain and D.W.J. Holmes, "The Digital Cordless Telecommunications Common Air Interface," *British Telecom Technical Journal*, Vol. 8, No. 1, pp 12-18, Jan. 1990.

[Tia90] Telecommunications Industry Association, *EIA/TIA Interim Standard, Cellular System Dual-Mode Mobile Station - Base Station Compatibility Standard, IS-54*, Telecommunications Industry Association, Washington, D.C. May 1990.

[Trw94] Telecommunication Reports, *Wireless News*, Vol. 4, No.3, Feb. 10, 1994.

[Wal88] J. Walfisch and H.L. Bertoni, "A Theoretical Model of UHF Propagation in Urban Environments," *IEEE Transactions on Antennas and Propagation*, Vol. 36, pp. 1788-1796, Dec. 1988.

[War92] *ITU World Administrative Radio Conference* (WARC'92), Torremolinos, Spain, Feb.3-Mar. 3, 1992.

[Wel89] R.A. Welch, "Desktop mapping with personal computers", *Photogrammetric Engineering and Remote Sensing*, Vol. 55, No. 11, pp. 1651-1662, Nov. 1989.

[Wel90] R.A. Welch, "3-D terrain modeling for GIS applications", *GIS World*, Vol. 5, No. 5, pp. 26-30, Oct/Nov 1990.

[Wes91] James Westervelt et al., "Introduction to GRASS 4," *GRASS 4.0 Users' Manual*, U.S. Army Construction Engineering Research Laboratory, Illinois, 1991.

[You79] W.R. Young, "Advanced Mobile Phone Service: Introduction, Background, and Objectives," *Bell System Technical Journal*, Vol. 58, No. 1, pp. 1-14, Jan. 1979.

[You90] Douglas Young, et al, *The X Window System: Programming and Applications with Xt, OPEN LOOK Edition*, Prentice Hall, 1990.

Appendix A. AutoCAD DXF Interface

A.1.1 Introduction

AutoCAD maintains a drawing database in which the drawing files are stored in a a compact format. Although this format cuts down on disk storage requirements, it changes significantly with new software releases. Since this format subject to many changes, it is not explicitly documented and it is recommended that programs that access the AutoCAD database, do not read or write this format.

In order to assist the interchange of drawings between different software packages, a Drawing Interchange Format (DXF) has been defined. The advantage of using the DXF format is that all implementations of AutoCAD have the ability to import and export data in this format. Additionally, they can be easily translated to the formats of other CAD software systems or submitted to other programs, like SISP, for specialized use.

This appendix provides an overview of the structure of AutoCAD DXF files and describes the salient features of the format. The information presented in this appendix is intended to serve as a programmers' reference for interpreting the contents of building data files obtained for SISP. It may be convenient to use a small DXF file when reviewing the information presented in this appendix. An exhaustive discussion of the AutoCAD DXF file format is outside the scope of this thesis. A complete definition of the format can be found in [Acad90].

A.1.2 DXF Files

This section provides an introduction to DXF files. DXF files are standard ASCII text files. AutoCAD can also produce or read a binary form of the full DXF file. The ASCII version serves to circumvent portability issues, since it is system independent. In general, since no assumption can be made about the source of the data files, the standard ASCII text file format is adopted for data interchange. The deficiencies of this technique are that the data files can be extremely large, and of a limited accuracy.

DXF files can be generated from existing drawings by means of the drawing editors' *dxfout* command. The format of this command is:

dxfout file_name

This command obtains the value of the precision, desired for floating point numbers, from the user. Using the *entities* option of the command, partial DXF files containing user selected objects can be created.

A.1.3 Overview of the File Structure

DXF files are standard ASCII text files with specially formatted text. The overall organization of the DXF file is listed below.

1. **HEADER** section: General information about the drawing is found in this section of the DXF file. Each parameter has a variable name and an associated value.
2. **TABLES** section: This section contains definitions of named items such as:

- Linetype table (*LTYPE*)
- Layer table (*LAYER*)
- Text Style table (*STYLE*)
- View table (*VIEW*)
- User Coordinate System (*UCS*)

3. **BLOCKS** section: This section contains Block Definition entities describing the entities comprising each Block in the drawing.
4. **ENTITIES** section: This section contains the drawing entities, including any Block References.
5. END OF FILE section: This section does not contain any data and serves to mark the end of the data section.

If the *entities* option of the *dxfout* command is used, the resulting DXF file will only contain the *ENTITIES* and *END OF FILE* section. Moreover, the *ENTITIES* section will reflect only the objects that are selected for output by the user.

A DXF file comprises multiple groups, each of which occupies two lines in the DXF file. The first line of a group is the group code which is a positive, non-zero integer. Group codes are used to indicate the general use of the group and the type of value the group can contain. The second line of the group is the group value in a format dependent on the group code. The specific assignment of group codes depends upon the item being described in the file. A list of typical group codes and brief descriptions is given in Table 1-1.

Table A.1-1. Group codes commonly present in DXF files

Group Codes	Value type
0	Identifies the start of an entity, table or file separator
1	Primary text value for an entity
2	A name; Attribute tag, Block name etc.
8	Layer name
10	Primary X coordinate (start point of a Line entity)
11-18	Other X coordinates
20	Primary Y coordinate. 2x values always correspond to 1x values and are successive entries in the file.
21-28	Other Y coordinates
30	Primary Z coordinate. 3x values always correspond to a 1x value and are successive entries in a file
31-37	Other Z coordinates
39	An entity's thickness, if nonzero
50-58	Angles
62	Color number
66	"Entities follow" flag

Table A.1-1. Group codes commonly present in DXF files

Group Codes	Value type
70-78	Integer values, such as repeat counts, flag bits
999	Comments

The type of value that each group code supplies is known from the group code. Thus any program can read the value following any group code without knowing the particular use of the group, in the data element in the file. The appearance of values in the DXF file is not affected by any settings of the AutoCAD *units* command. Thus, coordinates are always represented as decimal numbers, and angles are always represented in decimal degrees with zero degrees in a direction due east of the origin.

Variables, table entries, and entities are described by a group that introduces the item giving its type and/or name, followed by multiple groups that supply the values associated with the item. In addition, special group codes are used for file separators such as markers for the beginning and end of sections, tables, and the file itself. Entities, table entries and file separators are always introduced with a group code 0 that is followed by a name describing the item.

A.1.4 DXF File Sections

The DXF file is subdivided into four sections. File separator groups are used to delimit these file sections.

A.1.4.1 HEADER section

The *HEADER* section of the DXF file contains the settings of variables associated with the drawing. These variables are set with various drawing editor commands and are essentially the information obtained using the *status* command in the drawing editor. Each variable is specified in the header section by a group code 9, giving its name, followed by groups that supply its value.

A.1.4.2 TABLES section

The *TABLES* section contains several tables, such as the *LTYPE* table, *LAYER* table, etc., each of which in turn contains a variable number of table entries. The order of the tables may change, but the *LTYPE* table always precede the *LAYER* table. Each table is introduced with a group code 0, with the label *TABLE*. This is followed by a group code 2 identifying the particular table (*VPORT, LTYPE, LAYER, STYLE, VIEW, DIMSTYLE, UCS* or *APPID*) and a group code 70 that specifies the maximum number of tables entries that may follow.

Following this header for each table are the table entries. Each table item consists of a group code 0 identifying the item type (same as table name, e.g., *LTYPE* or *LAYER*), a group code 2 giving the name of the table entry, a group code 70 specifying flags relevant to the table entry (defined for each following table), and additional groups that give the value of the table entry. The standard flag values in the table apply to all entries in the table. The end of each table is indicated by a group code 0 with the value *ENDTAB*.

A.1.4.3 BLOCKS section

The *BLOCKS* section of the DXF file contains definitions for all the blocks, which are groups of entities classified using some basis. This section contains the entities that make up the *BLOCKS* used in the drawing. The format of the entities in this section is the same as the format in the *ENTITIES* section of the file. All entities in the *BLOCKS* section appear between the *BLOCK* and *ENDBLK* entities. Block definitions are never nested. References to blocks external to the drawing file are written into the DXF file like any other block definition, except that they also include a text string, of group code 1, specifying the path and file name of the external reference.

A.1.4.4 ENTITIES section

Entity items appear in both the *BLOCKS* and *ENTITIES* sections of the DXF file. Some groups that compose the definition of an entity always appear, some are optional, and some appear only if they differ from their default values. No assumptions can be made regarding the order in which the groups describing an entity occur in a DXF file.

Each entity begins with a group code 0 identifying the entity type. The end of group definitions that compose an entity is indicated by the next group code 0 that occurs in the file. Every entity contains an 8 group that gives the name of the layer on which the entity resides. Each entity may have an elevation, thickness, linetype or color information associated with it.

As stated earlier, DXF files are complete descriptions of the drawing database. New groups are bound to be added with newer releases to accommodate additional features. Therefore, DXF processing programs cannot make any assumptions about the order in which the groups occur. Moreover, the programs can ignore groups that it cannot resolve and proceed with the conversion.

The names of entities commonly found in DXF files are given in Table 1-2.

Table A.1-2. Common entity types

Entity type	Description
LINES	Lines can be drawn with various dot-dash linetypes. When drawing a line segment a 2D or 3D coordinate can be provided.
TEXT	Text can appear in various fonts, with any size and orientation that is needed.
ARCS & CIRCLES	Arcs and circles can be drawn with various dot-dash linetypes.
BLOCKS	Blocks are compound entities formed from groups of other entities
ATTRIBUTES	Attributes attach constant or variable text information to each instance of a Block.
POLYLINES	Polylines are 2D connected line and arc segments, with optional dot-dash linetypes, width and taper.
3D *POLYLINES*	3D Polylines are fully general three-dimensional objects composed of straight-line segments.
POLYFACE MESHES	Polyface meshes are general polygon meshes of arbitrary topology. A polyface mesh is created by defining vertices and faces composed of those vertices.

The group codes used to indicate the type of entities were described in the preceding paragraphs. The paragraphs that follow discuss the entity description and the groups it comprises. Many of the entities include *flag* groups. These integer codes, 6x and 7x groups, encode various pieces of information regarding the entity and are specific to the entity type. In the descriptions that follow, the term bit-coded means that the flag contains various true/false values coded as the sum of the bit values given.

A.1.4.5 POLYLINES entity

The *flag* groups and their description are given in Table 1-3

Table A.1-3. Group codes for *POLYLINE* entities and their description

Flag	Description
66	Vertices follow flag
70	Polyline flag (optional, 0)
40, 41	Default starting and ending width (optional, 0)
71, 72	Polygon mesh M and N vertex counts (optional, 0)
73, 74	Smooth surface M and N densities (optional, 0)
75	Curves and smooth surface type (optional, 0)

The *vertices follow flag* (group code 66) is always 1, indicating that a series of *VERTEX* entities is expected to follow the *POLYLINE*. This series is terminated by a *SEQEND* entity. The *polyline flag* (group code 70) is a bit encoded field which can contain the values shown in Table 1-4.

Table A.1-4. Group 70 bit codes for the *POLYLINE* entity

Flag bit value	Description
1	Closed Polyline (or a polygon mesh closed in the M direction)
2	Curve fit vertices added
4	Spline fit vertices added
8	3D Polyline

Table A.1-4. Group 70 bit codes for the *POLYLINE* entity

Flag bit value	Description
16	3D polygon mesh
32	Polygon mesh closed in the N direction
64	Polyline is a polyface mesh

Following the *POLYLINE* entity is a series of *VERTEX* entities that specify the vertex coordinates that compose the entity. The vertex indices are determined by the order in which they appear within the *POLYLINE*, the index starting with 1.

A.1.4.6 VERTEX entity

The various groups relevant to the entity are given in Table 1-5.

Table A.1-5. Group codes for the *VERTEX* entity and their description

Group codes	Description
10, 20, 30	X, Y, Z location
40, 41	Starting and ending width (optional, 0)
42	Bulge (optional, 0)
70	Vertex flags (optional, 0)
50	Curve fit tangent direction (optional)

The group codes 10, 20, and 30 specify the X, Y, and Z coordinates of the vertex. The *vertex flag* (group code 70) is used to indicate the type of the vertex. The interpretation of the bit-coded *VERTEX* flag (group code 70) is shown in Table 1-6.

Table A.1-6. *VERTEX* flag codes and their description

Flag bit value	Description
1	Extra vertex created by curve fitting
2	Curve fit tangent defined for the vertex
4	Unused (never set in DXF files)

Table A.1-6. *VERTEX* flag codes and their description

Flag bit value	Description
8	Spline vertex created by spline-fitting
16	Spline frame control point
32	3D Polyline vertex
64	3D polygon mesh vertex
128	Polyface mesh vertex

A.1.4.7 SEQEND

No fields. This entity marks the end of vertices for a *POLYLINE*.

A.1.5 Writing DXF Interface Programs

On examining DXF files manually, they seemingly contain an overwhelming amount of information. However, the DXF file has been designed to be easy to process by programs and not manually. The format is constructed in such a way that it makes it easy to ignore information that is not necessary. The most important consideration in the DXF interface programs should be to provide the capability to handle the groups in any order and also ignore group codes that are not necessary, or that cannot be resolved.

This appendix introduces the AutoCAD DXF file format. The building data for SISP is obtained as AutoCAD DXF files. In these files, the buildings are represented as two dimensional polyline entities with the thickness of the polyline used to denote building heights. The appendix provides information necessary for the interpretation of such files and for writing programs that import the entities into the SISP vector database.

Appendix B. Function Reference

Overview

This appendix provides a reference page for important functions in the SISP raster graphics library. The reference pages provide all the details necessary for invoking each routine such as its prototype, the arguments, the returned values, and errors that it might generate.

A description of the general format of the function reference pages is provided in Figure 1.

Name

Function Name - brief description of the function.

Synopsis

The Synopsis section presents the calling syntax for the routine, including declarations of the arguments and the return type.

Arguments

The Arguments section contains a description of each of the arguments used by the function.

Description

The Description section describes what the function does.

Figure B-1. General format of the reference pages. A brief description of the contents of each section has been provided.

Name

AddColorRule -- Store RGB values corresponding to raster data values, in a linked list structure.

Synopsis

void AddColorRule (cat1, r1, g1, b1, cat2, r2, g2, b2, cp, cmin, cmax)

CELL cat1, cat2

int r1, r2, g1, g2, b1, b2

*struct _Color_Info_ *cp*

*int *cmin, *cmax*

Arguments

cat1	Lower limit of the interval of data values
r1	Value of red corresponding to cat1
g1	Value of green corresponding to cat1
b1	Value of blue corresponding to cat1
cat2	Upper limit of the interval of data values
r2	Value of red corresponding to cat2
g2	Value of green corresponding to cat2
b2	Value of blue corresponding to cat2
cp	A pointer to the _Color_Info_ structure in which the linked list is a member.
cmin	The minimum data value stored in the structure *Colors*
cmax	The maximum data value stored in the structure *Colors*

Description

This function is used to add elements to a linked list, which stores the color rules, for intervals in the range of data values present in the raster file. *cat1* and *cat2* are the limits of the intervals of the data values. *r1*, *g1*, and *b1* are the RGB values associated with *cat1*. *r2*, *g2*, and *b2* are the RGB values associated with *cat2*. This element is inserted in a linked list which is a member of the structure *cp*.

The range of data values in the raster file is split into intervals and this function is called for each of the intervals. The RGB values for any data value in this interval is obtained by interpolating between the RGB values of *cat1* and *cat2*.

Name

AllocNewImage -- Function to allocate memory for the XImage data.

Synopsis

int AllocNewImage (void)

Arguments

None

Description

This function performs the initialization required to display raster images. The width of the image to be displayed is calculated. The function allocates memory for the *D_to_A_tab*. The *D_to_A_tab* provides the conversion between the display coordinates and the data array indices, utilizing the conversion factors computed by the routine *ViewingTransforms()*. The function calls *XGetWindowAttributes()* to obtain information about the visual used to display the raster image. It allocates memory to contain the pixel values of one row of the displayed image. It creates an *XImage* structure, which is used to display the raster image. This function is called by the function *CellDraw()*. The memory allocated in this function is freed in the function *DestroyOldImage()*.

Name

CellDraw -- Function to read in the data from the raster data value file and display the image.

Synopsis

int CellDraw (pathname, mapname, colors, zoom)

*char *pathname, mapname'*

*struct Colors *colors*

int zoom

Arguments

pathname	Pathname of the top level database directory.
mapname	Name of the raster data file.
colors	Structure containing color rules.
zoom	Flag to indicate if the operation is a zoom or not.

Description

This function reads in a raster data file and displays the raster image. This function performs the core function for displaying the raster data. The function is called by *PaintRaster()* to display the image. The routine reads in the header file associated with the raster data file. If the operation is a zoom operation in which the area to be displayed is explicitly specified by the user, the function does not overwrite the

global variable *G__* with the values read in from the header file. If the operation is not a zoom operation then the values read in from the header file are copied to the global variable *G__*.

The function *SetRasDisplay()* is called to setup the viewing transformation factors. These are used in translating the user coordinates into the display coordinates. The function checks if the variable *map_in_memory* is set. If the variable is not set, then the function allocates memory for the entire data file and reads in the data file using the routine *G_get_map_row_nomask()*. If the variable is set the data file is not read in again.

The function *AllocNewImage()* is called to allocate memory to hold the image data and to setup the array to perform the column to display width mapping. The raster image is displayed by calling the function *D_draw_cell()* for each row in the raster data file. The memory allocated for storing the image is freed with a call to *DestroyOldImage()*. The function returns 0 on success and a negative number on failure.

Name

ComposeFmtStr -- Plotting library function to format the string with which the data values in the set are displayed.

Synopsis

void ComposeFmtStr (fmt_str, tab_index)

*char *fmt_str*

int tab_index

Arguments

fmt_str	The format string that is passed to the *printf()* routine
tab_index	The index into a static table of format strings used to determine the appropriate value to use for a given data set.

Description

This function is used to compose the format string that is passed to the *printf()* function to display the data values in the plot. The *tab_index* value is used as an index into a static table of format strings, and is used to determine the appropriate value of the string for the given range of values. The value of this variable is determined by the function *NewComputeIntervals()*.

Name

CreatePixmaps -- Function to create pixmaps of different dimensions.

Synopsis

int CreatePixmaps (factor)

int factor

Arguments

factor	Zoom factor. The dimension of the pixmap created is *factor*default window dimensions*.

Description

This function creates the pixmaps used by the application. This function is called from *WinInit()* and it creates a pixmap with factor 1. This pixmap is created by default everytime a new window is initialized. This is used to redraw the image on to the visible window for normal drawing operations.

During a zoom operation, the function is called with a factor > 1. Currently only factors <= 4 are supported. Attempting to create pixmaps bigger than this generates Xlib errors. The function does the book-keeping for these pixmaps. The details regarding the active pixmaps are maintained in a table *zoomchoice*. *zoomflag* is the variable that indicates the pixmap that is currently active, and *zoomindex* is the variable that indicates the next location in the zoomchoice table that is free for use. The idea with the book-keeping is to maintain *ZOOM_ARR_SIZE* pixmaps active at any point of time. The pixmap that was allocated the earliest is marked for re-use. The pixmap of factor 1 is never released and thus the slot 0 in table cannot be re-used.

Each time the function is invoked, the table is checked to see if the pixmap of the same dimension is already active. If this is true, then the index into the table for this pixmap is returned by the function. If the pixmap is not active, a new pixmap needs to be created. The function creates the new pixmap, updates the details of the pixmap in the table and returns 0. In case of error this function returns a negative number.

Name

D_check_colormap_size -- Function that is used to check if the number of colors needed to display the data is less than the number of colorcells in the display.

Synopsis

int D_check_colormap_size (color_min, color_max, ncolors)

CELL color_min, color_max

*int *ncolors*

Arguments

color_min	The minimum value in the range of data values
color_max	The maximum value in the range of data values
**ncolors*	The number of color values that is used.

Description

This function is used to used to determine and fix the number of colors that will be used to display the data values. If the range of data values exceeds the number of colorcells in the colormap (typically 256 in an eight plane display), then the number of colors that are used to display the data is limited to the number of colorcells in the colormap. If the range of data values is less than the number of colors in the colormap, the number of colors used to display the data is the range of data and two more for the colors white and black. This function returns a value of 0 if the range of data exceeded the number of colorcells and 1 if the range was lesser.

Name

DeleteAntenna-- Function to delete antenna icons at locations specified by the user.

Synopsis

void DeleteAntenna (x_coord, y_coord, saved_image)

int x_coord, y_coord

*XImage *saved_image*

Arguments

x_coord	The x coordinate of the location in display coordinates
y_coord	The y coordinate of the location in display coordinates
saved_image	The *XImage* structure used to store the underlying image when the antenna icon is located.

Description

This function is used to delete an antenna icon located in the display window and restore the image that was saved when the antenna was deleted. The *saved_image* variable contains the image that needs to be restored. The *x_coord* and *y_coord* can be in display coordinates, since the edit operations will be done at the same display resolution at which the icons were located. However, if the display resolution is changed after the icons have been located, the icons are relocated on the new image and thus the underlying image is obtained once again.

Name

DestroyOldImage-- Function to free memory allocated for the *XImage* structure.

Synopsis

int DestroyOldImage (void)

Arguments

None

Description

This routine is used to free the memory allocated for displaying the raster data. The variable *grimage* is used to store the data associated with the *XImage* structure used to display the raster image. The *D_to_A_tab* is the array that is used to perform the conversion between the display coordinates and the data array indices. The memory allocated for grimage and the *D_to_A_tab* is freed by this routine.

Name

DispRasterRow --Function to display the raster image corresponding to one data row.

Synopsis

void DispRasterRow (ncols, nrows, array)

int ncols, nrows

*unsigned int *array*

Arguments

ncols	The number of columns in the data file.
nrows	The number of rows in the display window that the data row corresponds to.
array	Buffer containing the data values.

Description

This function is used to display a row of the raster image in the display window. The *ncols* variable specifies the number of columns in the data array and the *nrows* variable specifies the number of rows the data values have to be displayed for. The memory is allocated for the *XImage* structure and the image data by the function *AllocNewImage()*. The routine *get_fixed_color_array()* called to perform a lookup between data values and pixel values. The pixel values are stored in the array *imdata*, after performing the conversion between the data array values and the display coordinates. The Xlib routine *XPutImage()* is used to display the image in the active display window.

Name

DrawXLabel -- Plotting library function to display the x-axis label for the plots.

Synopsis

void DrawXLabel (plot_widget, FInfo, xlab_str)

Widget plot_widget

*XFontStruct *FInfo*

*char *xlab_str*

Arguments

plot_widget	Widget identifier of the display window
FInfo	The structure which specifies the characteristics of the font used.
xlab_str	The text string for the x-axis label

Description

This function is used to the x-axis label for the plots..

Name

DrawYLabel -- Plotting library function to display the y-axis label for the plots.

Synopsis

void DrawYLabel (plot_widget, FInfo, labstr)

Widget plot_widget

*XFontStruct *FInfo*

*char *labstr*

Arguments

plot_widget	Widget identifier of the display window
FInfo	The structure which specifies the characteristics of the font used.
labstr	The text string for the y-axis label

Description

This function is used to the y-axis label for the plots. Since the Xlib library does not support rotation of images, this function rotates the text string supplied to be used as the y-axis label.

Name

DumpWindow -- Function to save the terrain elevation information associated with a selected region of the displayed image.

Synopsis

int DumpWindow (srcpath, srcfile, despath, destfile, option)

*char *srcpath, *srcfile, *destpath, *destfile*

int option

Arguments

srcpath	Path of the source terrain elevation file
srcfile	Name of the terrain elevation data file (compressed format).
destpath	Path of the destination file
destfile	Name of the terrain elevation file to be created (uncompressed format).
option	Currently unused.

Description

This function is used to save the terrain elevation information for the region displayed in the display window. The format of the saved file is the GRASS 4.0 uncompressed raster date file format. The routine is provided with the name of the terrain elevation file which is currently displayed and the name of the uncompressed raster data file that has to be created. The option argument is currently unused but can be used in the future for specifying the format in which the data is to be saved.

Name

FindFileSet -- Search for different database files associated with the given file name.

Synopsis

int FindFileSet (pathname, filename)

*char *pathname, filename*

Arguments

pathname	Name of the top level database directory, under which search is performed.
filename	Name of file to be searched for.

Description

This routine is used to check the existence of all the database files associated with a raster data file. Currently this function only checks for the existence of the header file in the directory *cellhd* under the pathname specified by the user. The function checks for the existence of the file by using the system call *access()*. The file is checked to see if it exists and if it is accessible for reading. The function returns 0 on success and -1 on error.

Name

FindMiscFile -- Search for the file name specified.

Synopsis

int FindMiscFile (fullmapname)

*char *fullmapname.*

Arguments

fullmapname The name of the file specified with an absolute path to the file

Description

This function is used to check the existence of any file specified by the user. The function calls the system call *access()*, to determine if the file specified by fullmapname exists and if it is accessible. The *fullmapname* variable contains the entire path to the file and the name of the file.

Name

FindRange -- Plotting library function to determine the range of the data set.

Synopsis

int FindRange (datasize, dataval, limit)

int datasize

*FPTYPE *dataval*

*FPTYPE *limit*

Arguments

datasize	The size of the data set
dataval	Pointer to the data set
limit	The variable in which the range of values is returned.

Description

This function is used to determine the range of the values in the data set.

Name

FindTextStrDim -- Plotting library function to determine the height and width of text string. Could be used for labeling the plots.

Synopsis

void FindTextStrDim (FInfo, in_string, str_hite, str_wid)

*XFontStruct *FInfo*

*char *in_string*

*int *str_hite*

*int *str_wid*

Arguments

FInfo	The structure which specifies the characteristics of the font used.
in_string	The text string whose dimensions are to be determined
str_hite	Computed height of the string
str_wid	Computed width of the string

Description

This function is used to determine the dimensions of a text string in any font. This could be used to determine the location of labels in plots.

Name

G__check_format -- Function to check the format of a cell file.

Synopsis

int G__check_format (fd)

int fd

Arguments

fd	File descriptor of open cell file

Description

This function is used to check the format of the cell data file. The file can be in either the uncompressed format or the compressed format. Formats earlier than the GRASS-3.0 format are not supported. The format of the file is indicated in the header file. This information is available in the file control buffer for this file. This is a global variable.

For files in the compressed format, the function allocates memory to store the row offset table. It calls the function *G__read_row_ptrs()* to read the offset table from the cell file. These operations are not performed if the file is in the uncompressed format.

This function returns -1 in case of error.

Name

G__read_row_ptrs -- Function to read the row offset table from the cell file.

Synopsis

int G__read_row_ptrs (fd)

int fd

Arguments

fd	File descriptor of open cell file

Description

This function is used to read the row offset table from the cell file. This function stores the offset table in the file control buffer, and assumes that memory has been allocated for this. This function should be called from within G__check_format, where this memory is allocated.

The row offset table is stored at the start of the cell data file. The offset of each row is converted to binary and stored in the file. This function reads the offset values, converts them to decimal numbers and stores it in the array allocated for the purpose.

This function returns -1 in case of error and 1 otherwise.

Name

GetBuildingHeight -- Obtains the building height information corresponding to the position of the mouse pointer.

Synopsis

int GetBuildingHeight (x, y, height)

int x

int y

*int *height*

Arguments

x	Coordinate of the mouse pointer along the x-axis.
y	Coordinate of the mouse pointer along the y-axis.
height	The height of the building corresponding to the given coordinates.

Description

This function is used to provide continuous height tracking of buildings. The routine obtains the coordinates of the mouse pointer and uses it to determine the elevation information of the building corresponding to that position.

Name

InitColorTableFixed -- Allocate colormap used by the application

Synopsis

Colormap InitColorTableFixed (void)

Arguments

None

Description

This function is used to allocate the colormap which is used by the application. The number of display cells available in the server is determined with a call to *DisplayCells()*. *Red*, *Grn* and *Blu* are static arrays used to store the color lookup table. These are iteratively filled with values of red, green and blue, such that equal levels of red, green, and blue are possible. An attempt is made to allocate the colors necessary for the application, from the default colormap, with a call to *XAllocColor()*. If all the desired colors cannot be allocated from the default colormap, a new colormap, is allocated with a call to the function *XCopyColorMapFree()*. The pixel values returned for each combination of red, green and blue returned by the function *XAllocColor()* is stored in the array *xpixels*. The function returns the identifier for the colormap created and initialized.

Name

LoadDefaultImages -- Function to load bitmaps and create icons for display.

Synopsis

int LoadDefaultImages (void)

Arguments

None

Description

This function is used to load bitmaps to create icons for display. This function is used to create the default icons, which need to be displayed when the user needs to interactively locate antennas. The bitmaps to be loaded by the application as default, are specified at compile time. One file for the transmitter and one file for the receiver are specified. The function calls *SCreateImageFromXbm()* to load the bitmaps specified as default. This function returns 0 on success and -1 if either of the bitmaps cannot be loaded.

Name

LocateAntenna -- Function to locate antenna icons at locations specified by the user.

Synopsis

int LocateAntenna (in_x, in_y, mode, antpick)

double in_x, in_y

int mode

antennaType antpick

Arguments

in_x	The x coordinate of the position selected by the user
in_y	The y coordinate of the position selected by the user
mode	The mode specifies whether the coordinates provided are display coordinates or UTM coordinates
antpick	An enumerated data type to indicate the type of antenna icon the user wants to locate at that position.

Description

This function is used to locate an icon at the location specified by the user. The input to the function can be display coordinates or it can be UTM coordinates. If UTM coordinates are provided the position in the display window is determined by applying the coordinate transformation factors. This mode is utilized when redrawing the icons after a zoom operation.

The image that is overlaid by the icon is saved. This is restored if the user deletes the antenna located at that point. The image for the icon to be displayed is obtained from an *ImageTable* that stores the icon image for various antenna types. This table has to be initialized by the user interface module. This function returns 0 on success and -1 on error.

Name

LookupPixelValue -- Returns the pixel value corresponding to a triplet of *red*, *green* and *blue* color values.

Synopsis

unsigned long LookupPixelValue (red, green, blue)

int red, green, blue.

Arguments

red	Red color value in the triplet
green	Green color value in the triplet
blue	Blue color value in the triplet

Description

This function is used to return the pixel value corresponding to a triplet of *red*, *green* and *blue* values. The *red*, *green* and *blue* values passed to the function are used as indices into the static arrays *Red*, *Grn* and *Blu*. These arrays are initialized by the function *InitColorTableFixed()*. The sum of the values returned from the individual *Red*, *Grn* and *Blu* lookups are summed to yield the index into the *xpixel* array. The function returns the value returned from the *xpixel* array.

Name

NewComputeIntervals -- Plotting library function to determine the number of intervals into which the data should be partitioned for display purposes.

Synopsis

int NewComputeIntervals (limit. maxitvl, intvl_tab_index)

*FPTYPE *limit*

int maxitvl

*int *intvl_tab_index*

Arguments

limit	The upper and lower limits of the data set
maxitvl	The maximum number of permissible intervals
intvl_tab_index	The index into the interval table

Description

This function is used to determine the number of intervals into which the data set has to be partitioned for display purposes. The maximum number of intervals sets the upper limit for the number of intervals. The function determines the range of values in the data set and uses a static table of values to determine the nearest value to which the limits should be rounded-off. It then determines the number of intervals into which this new range is partitioned. The *intvl_tab_index* refers to the index in this table and is used by other routines. The number of intervals is returned by the function. However, if the range of data values in the set is zero, a default value of 5 is returned as the number of intervals.

Name

PaintRaster -- Function to display the raster image.

Synopsis

int PaintRaster (cat1, r1, g1, b1, cat2, r2, g2, b2, cp, cmin, cmax)

*char *pathname, mapname*

int filter_switch

Arguments

pathname	Name of the path in which the fie is located.
mapname	Name of the bitmap file to be read.
filter_switch	Flag to indicate if the displayed raster image. needs to be flattered.

Description

This function displays a raster image in a display window. The function checks for the existence of the data file and related files specified by the user as the pathname and mapname arguments. It then checks for the presence of a color file which it reads in. If the color file does not exist, a set of color rules are formed which associate data values with RGB values.

The routine *SetColors()* is called which checks if the number of colors required to display the raster image is less than the number of hardware colors. If this is true then the routine performs the association of RGB value to pixel value for every data value within the range of the values in the raster data file. If the number of colors required to display the raster data is greater than the number of hardware colors, then color translation tables are formed and the RGB values in this table are associated with the pixel values.

The function *CellDraw()* is called to display the raster map. The routine reads the header file for the raster data, initializes the viewing transformation factors, and reads the data file into memory. The routine allocates memory to store the data associated with the raster image to be displayed. The routine obtains the pixel values corresponding to the data values, for each row of data and displays the raster image.

The function returns 0 on success and a value less than 0 on error.

Name

PaintValueScale -- Function that is used to display the legend for the image displayed.

Synopsis

int PaintValueScale (scale_widget, path, map, filter_switch, display_units)

Widget scale_widget

*char *path, *map*

int filter_switch

unitType display_units

Arguments

scale_widget	The widget identifier of the window in which the legend is displayed
path	The directory path in which the file is located
map	The name of file which contains the data values for which a legend is generated
filter_switch	Flag to indicate whether color filtering needs to be performed
display_units	Enumerated data type to indicate the system of units

Description

This function is used to generate a legend for the image shown in the display window. The function takes the directory path and the name of the file whose data values have to be used for generating the legend. The values are read from the range file corresponding to the data file and the range of values is determined. The range is divided into 10 equal intervals and RGB values are associated with each of the data values. The RGB values are then used to determine the pixel values to be displayed corresponding to each data value. The scale is drawn using the Xlib drawing primitive *XDrawRectangles()*. The pixel value corresponding to each data value is used to fill the rectangle and the data value it represents is displayed inside each rectangle. This function returns -1 on error and 0 otherwise.

Name

PlotInstPower -- Plotting library function to display the instantaneous received power and data similar to this (essentially any 2D rectangular plot).

Synopsis

int PlotInstPower(plot_widet, plot_no)

Widget plot_widget

int plot_no

Arguments

plot_widget	Widget identifier of the display window
plot_no	Index used to denote the array to be chosen for display. This is only a temporary fix and the interface to this function needs to be modified at the time of integration.

Description

This function is used to display any two dimensional plot. It calls a set of routines, which are part of the plotting library, to accomplish the function. It displays the data values in a two dimensional plot, draws the axes, labels the plot and marks the intervals of data values along the axes.

Name

PlotPolarAxis -- Plotting library function to draw the axis for polar plots.

Synopsis

void PlotPolarAxis (plot_widget, intvl, limits, intvl_index)

Widget plot_widget

int intvl

*FPTYPE *limits*

int intvl_index

Arguments

plot_widget	Widget identifier of the display window
intvl	Number of intervals the data has been partitioned for display purposes
limits	The variable in which the range of values is stored.
intvl_index	Index used to determine the format in which the data values are displayed.

Description

This function is used to draw the axis for polar plots. The function draws concentric circles divided into sectors of 30 degrees. The sectors are labeled, and the *intvl_index* variable is used to determine the format of the displayed values from a static table.

Name

PlotRectAxis -- Plotting library function to draw the rectangular axis, for two dimensional plots..

Synopsis

void PlotRectAxis (plot_widget, xoffset, yoffset)

Widget plot_widget

int xoffset

int yoffset

Arguments

plot_widget	The widget identifier of the window in which plot is displayed
xoffset	The x coordinate offset from the origin of the window
yoffset	The y coordinate offset from the origin of the window

Description

This function is used to draw a rectangular axis for two dimensional plots. The *xoffset* and *yoffset* arguments specify the origin of the axis. The routine computes the height and width of the axis from these arguments.

Name

ReadCellHeader -- Function to read the header file associated with raster data files.

Synopsis

int ReadCellHeader (pathname, mapname, window)

*char *pathname, mapname'*

*struct Cell_head *window*

Arguments

pathname	Pathname of the top level database directory.
mapname	Name of the raster data file.
window	Structure pointer to store the raster data header information.

Description

This function is used to read the header file associated with a given raster data file. The header file is located in a directory, *cellhd*, under the top level database directory. Prior to calling this routine, existence of the raster file is checked and hence the check to see if the file exists is not performed.

The function calls *G__read_Cell_head()* to read the header file. The variable *G__.window_set* is set to indicate that the variable containing the header has been initialized.

Name

ReleasePixmaps -- Function to release pixmaps and clear the entry in the pixmap table.

Synopsis

int ReleasePixmaps (index)

int index

Arguments

index	Index into the pixmap table to indicate the pixmap to be released.

Description

This function is called to release the pixmap in the location, pointed to by *index*, in the pixmap table. The pixmap is released with a call to *XFreePixmap()*. The variables in the table corresponding to the old entry are reset. If the index is greater than *ZOOM_ARR_SIZE*, all the pixmaps in the table are released and the table is initialized.

Name

Reset_colors -- Function that sets up a table associating pixel values and the data values.

Synopsis

int Reset_colors (min, max, red, grn, blu)

int min, max

*unsigned char *red, *grn, *blu*

Arguments

min	The minimum value of the range of colors needed
max	The maximum value of the range of colors needed
red	Red values for the data
grn	Green values for the data
blu	Blue values for the data.

Description

This function invokes the functions *LookupPixelValue(),* which performs the association between RGB values and pixel values from the colormap. It also invokes the function *assign_fixed_color()* which is used to store the association between data values and pixel values in memory. The functions are called iteratively till all the data values in the range have pixel values associated with them.

Name

RoundLimits -- Plotting library function to round off the limits of the data set to the nearest whole number value.

Synopsis

FPTYPE RoundLimits (limit, tickinc)

*FPTYPE *limit*

FPTYPE tickinc

Arguments

limit	The range of values in the data set
tickinc	The value to which the limits of the data set is round off

Description

This function is used to round off the limits of the data set to the nearest value specified in the variable *tickinc*. The value to which the limits are rounded off are determined by the range of values in the data set. The function returns the range using the rounded off limits.

Name

SCreateImageFromXbm -- Create an *XImage* from the bitmap file.

Synopsis

int SCreateImageFromXbm (file_name, color_name, iptr)

*char *file_name, *color_name*

*XImage **iptr*

Arguments

file_name	Name of the bitmap file.
color_name	Color in which the icon is to created, as specified by the user.
iptr	Pointer to the location in the table used to store icons.

Description

This function is used to create an *XImage* structure from a bitmap file. This function is used to read in a bitmap file (created using a standard bitmap editor), convert it into an *XImage* structure and store it in an array. The bitmap file resembles a segment of a C program. The format of the bitmap file is given in Chapter-3. The standard Xlib functions assume that the bitmap file has been included during compile time. Hence this routine is needed to create an *XImage* from a bitmap file specified at run time.

The function reads in the values in the bitmap file. The routine assumes that the bitmap files are of dimension 10x10. The color to be associated with this icon is specified in the variable color_name. The pixel value corresponding to the color_name is obtained by a call to the Xlib function *XAllocNamedColor()*. The Xlib function *XCreatePixmapFromBitmapData()* is used to create a pixmap from the bitmap data that was read in. The background and foreground values specified by the user is used to create the pixmap. The *XImage* structure is obtained by calling the function *XGetImage()* to obtain the image from the pixmap created. The *XImage* structure is stored in an array, which can be accessed by the user. The pixmap created using the bitmap data, is freed. This function returns 0 on success and -1 on error.

Name

SetColors -- Function to setup the color rules for a raster image that needs to be displayed.

Synopsis

void SetColors (colors, filter_switch)

*struct Colors *colors*

int filter_switch

Arguments

colors	Structure containing color rules.
filter_switch	Flag to indicate whether the displayed image is filtered.

Description

This function is called every time a raster image needs to be displayed. This function calls *D_check_colormap_size()* to determine if the number of colors required to display the raster image is greater than or lesser than the number of hardware colors.

If the colors required to display the raster image is less than the number of colors in the hardware colormap, the variable *fixed* is reset. For all the values in the data file, the pixel values are determined and stored in a table. The function *G_lookup_colors()* is called for this purpose. The RGB values corresponding to the data values is determined. The function *Reset_colors()* is called to make the association between the pixel and RGB values.The table thus contains entries of a pixel value corresponding to each data value.

If the colors required to display the raster image is less than the number of colors available in the hardware colormap, the variable fixed is set. A color translation table is setup. The pixel values for each of the entries in the table is determined.

Name

ViewingTransforms -- Function that sets up the coordinate transformation factors that are used by the display routines.

Synopsis

int ViewingTransforms (window, t, l, b, r)

*struct Cell_head *window*

int t, b, l, r

Arguments

window	The structure that contains the coordinate information and the number of rows and columns of data
t	Represents the top of the window, initialized to 0
b	Initialized to the height of the display window
l	Represents the left of the window, initialized to 0
r	Initialized to the width of the window.

Description

This function is invoked by the *CellDraw()* to set up the transformation coordinate factors. The transformation coordinates are used to translate between display coordinates and UTM coordinates, display coordinates and the number of data points, the UTM coordinates and the number of data points. This function is also used position the displayed image in the center of the display window.

Name

WinInit -- Function that is used to check if the number of colors needed to display the data is less than the number of colorcells in the display.

Synopsis

int WinInit (toplevel, canv, scales)

Widget toplevel, canv, scales

Arguments

top_level	The widget identifier of the top level display window
canv	The widget identifier of the display canvas.
scales	The widget identifier of the window in which the scale is displayed

Description

This function initializes the display window and sets up the appropriate attributes for the window. It calls the function *CreatePixmaps()* for creating the pixmap that is used as a backup for the display window. It initializes the colormap to be used by calling the function *InitColorTableFixed()*. The *Colormap* identifier returned by the function is specified as the colormap to be installed by the server. The function *LoadDefaultImages()* is invoked to load bitmaps for the default representations of transmitters and receivers.

This function is invoked once when a display window is created. This function returns -1 on error and 0 otherwise.

Name

ZoomIn -- Function to zoom in to a displayed raster image by providing the zoom factor.

Synopsis

int ZoomIn (pathname, mapname, filter_switch, factor)

*char *pathname, mapname*

int filter_switch, factor

Arguments

pathname	Pathname of the top level database directory.
mapname	Name of the raster data file.
filter_switch	Variable to indicate if the displayed image needs to be filtered.
factor	Zoom factor between 2 and 5.

Description

The function is used to display a zoomed in raster image, by providing the zoom factor. The zoom factor should be a value between 2 and 5. The raster data that needs to be displayed involves a zoom operation, hence the data files are not checked for existence.

The color file is read if it exists, else a color translation table is established. The pixmaps of the desired dimensions is allocated with a call to *CreatePixmaps()*. The function returns a new pixmap or returns a value to indicate that a pixmap of the same factor exists. If a pixmap of the same dimension agreed then the image is not redrawn but just copied from the pixmap to the display window.

If the pixmap created is new, then the function *CellDraw()* is called to redraw the image with the new zoom factor.

Name

ZoomInUTM -- Function to zoom into a UTM area specified by the user.

Synopsis

int ZoomInUTM (pathname, mapname, filter, north, south, east, west)

*char *pathname, mapname*

int filter

double north, south, east west.

Arguments

pathname	Pathname of the top level database directory.
mapname	Name of the raster data file.
filter	Structure containing color rules.
north	Northern most UTM coordinate.
south	Southern most UTM coordinate.
east	Eastern most UTM coordinate.
west	Western most UTM coordinate.

Description

This function is called when a sub-area of the displayed image needs to be displayed. The UTM coordinates passed to the function as parameters specifies the coordinates of the boundary of this sub-region. The UTM coordinates obtained are used to update the variable *G__*. The function *ZoomIn()* is then called with the value *UTM_ZOOM* in the value for the factor.

If the value of factor is *UTM_ZOOM*, the function *ZoomIn()* does not attempt to create a pixmap. It calls *CellDraw()* with the argument *zoom* set to 1. In *CellDraw()* since the zoom parameter is 1, the *G__* is not copied with information from the header file, since it contains the bounds of the sub-region of the monitor. The other display routines are called and a sub-region of a displayed.

Index of Functions

Function Reference 141
AddColorRule 142
AllocNewImage 143
CellDraw 144
ComposeFmtStr 145
CreatePixmaps 146
D_check_colormap_size 147
DeleteAntenna 148
DestroyOldImage 149
DispRasterRow 150
DrawXlabel 151
DrawYlabel 152
DumpWindow 153
FindFileSet 154
FindMiscFile 155
FindRange 156
FindTextStrDim 157
G__check_format 158
G__read_row_ptrs 159
InitColorTableFixed 160
LoadDefaultImages 161
LocateAntenna 162
LookupPixelValue 163
NewComputeIntervals 164
PaintRaster 165
PaintValueScale 166
PlotInstPower 167
PlotPolarAxis 168
PlotRectAxis 169
ReadCellHeader 170
ReleasePixmaps 171
Reset_colors 172
RoundLimits 173
SCreateImageFromXbm 174
SetColors 175
ViewingTransforms 176
WinInit 177
ZoomIn 178
ZoomInUTM 179

Vita

Subramanian T. Parameswaran (TPS) was born on December 4, 1969, in Madras, India. He received the B.E. degree in electronics & communication engineering from the University of Madras, India, in 1991. After graduation, he worked as a systems engineer with Nucleus Software, Madras, India, where he was involved in the development of India's first credit verification network. Since January 1993, he has been a member of the Mobile and Portable Radio Research Group at Virginia Tech, Blacksburg, VA. His graduate research has focussed on the development of a software tool for site specific propagation prediction and wireless communication system design. He begins employment at National Semiconductor Corporation, in Santa Clara, CA., in December 1994.

Mr. Parameswaran is a member of the IEEE and the ACM.

T.P. Subramanian