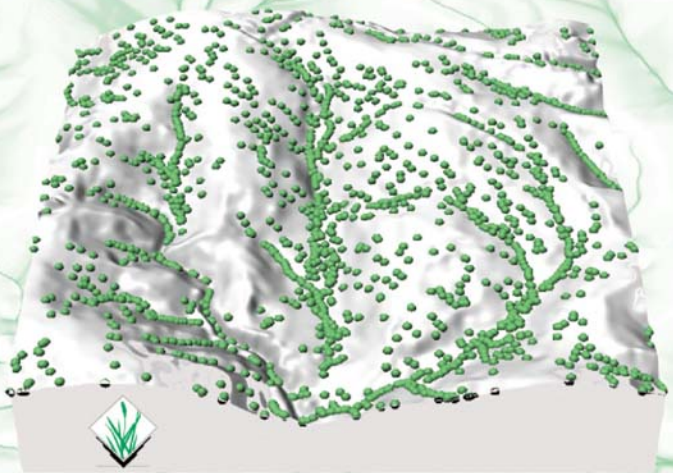
OPEN SOURCE GIS: A GRASS GIS Approach

Second Edition



Markus Neteler
and
Helena Mitasova

Open Source GIS:
A GRASS GIS Approach

OPEN SOURCE GIS:
A GRASS GIS APPROACH
Second Edition

MARKUS NETELER
ITC-irst – Centro per la Ricerca Scientifica e Tecnologica, Italy

HELENA MITASOVA
North Carolina State University, U.S.A.

KLUWER ACADEMIC PUBLISHERS
NEW YORK, BOSTON, DORDRECHT, LONDON, MOSCOW

eBook ISBN: 1-4020-8065-4
Print ISBN: 1-4020-8064-6

Created in the United States of America

Visit Springer's eBookstore at: http://ebooks.kluweronline.com
and the Springer Global Website Online at: http://www.springeronline.com

to our friends and to all
GRASS developers, present
and past

Contents

List of Figures

List of Tables

Foreword

William D. Goran, USA CERL

GRASS GIS software was developed in response to the need for improved analysis of landscape "trade offs" in managing government lands and the emerging potential of computer-based land analysis tools. During the last decades of the 20th century, government land managers in the U.S. (and across the world) faced increasing requirements from legislation and stakeholder groups to examine and evaluate alternative actions. To fulfill these new requirements, land managers needed new tools.

During this same era, computational capabilities wondrously improved. Tasks requiring days and months with paper and acetate overlays could be accomplished with this newly emerging geographic information technology within minutes. But even in the mid-1980s, GIS technology involved significant capital investment. Managers wanted to see results before they spent their limited funds on new technologies.

The U.S. Army Construction Engineering Research Laboratory (CERL) in Champaign, Illinois has the mission of developing and infusing new technologies for managing U.S. Department of Defense installations. These installations include millions of acres of lands needed for military training and testing. Other uses included wildlife management, hunting and fishing and forestry, grazing and agricultural production. Other priorities were added through legislation – such as protecting endangered species and habitats, protecting cultural sites, and limiting the on and off-post impacts of noise, ordnance, contaminants and sediments.

Military land managers were unable to cope with the challenge of examining proposed new actions (such as new weapon firing ranges or new vehicle training routes) without improved methods to gather, integrate and visualize their data and to examine alternative courses of action. Acquiring emerging propri-

etary technologies and digital data wasn't even a consideration – the cost was too high and the expertise required to learn, operate and manage the technology was beyond their resources.

Given this need, a group of then young researchers at CERL elected to develop their own set of initial landscape analysis tools. Initially, this in-house software development effort was designed to "bridge the gap" as commercial proprietary technology developed. The other costs involved in implementing GIS (acquiring data and hardware, learning GIS skills and computer maintenance skills) were so high; CERL decided that no-fee software could reduce the technology hurdle involved in implementing GIS. This proved to be true – and U.S. military installations were some of the first government managers to become active users of this new technology.

Once our efforts began, software development took on a life of its own. The Open Source code and Internet accessible software soon sparked the creative energies of numerous other organizations and individuals, and many began to use GRASS and contribute capabilities. At CERL, a small-scale skunk works project became the biggest and hottest program in the lab. Dozens of persons were employed developing new tools, building digital databases, assisting with complex applications and fielding the technology across the Department of Defense.

The needs we addressed drove the design criteria for GRASS. Because of the requirement to analyze alternative actions and to evaluate impacts of actions on continuous surfaces of differing elevations and vegetation and soil types, GRASS development was focused on raster analysis tools. Also, because of the need for digital and "real time" data, GRASS also incorporated remotely sensed image integration and analysis tools. At the time, this focus set GRASS apart from marketplace capabilities, which were primarily based on vector data and tools and did not include image analysis.

To nurture a "growing" GRASS community, CERL and other organizations established forums for sharing and contributing software. For several years, the lab (and lab partners) also offered newsletters, developed formal interagency partnerships (primarily with the U.S. Department of Agriculture and National Park Service) and held annual software user meetings. During the early 1990s, this GRASS community helped to initiate the Open GIS Foundation (now the Open GIS Consortium) as an international organization focused on advancing openness and interoperability for geospatial technologies.

But by the mid-1990s, many of the original military installation GIS users were switching to proprietary marketplace GIS technologies. In the intervening years, marketplace GIS vendors had added raster analysis tools, much like those in GRASS. Installation managers had become dependent on GIS, and were now willing to buy from the marketplace. Generally, the government is expected to buy off the marketplace, unless there are no comparable market-

place options. Plus, installation managers wanted GIS software just like the systems that were showing up in the offices of supporting contractors and local and state government offices across-their-fence lines. As a result, CERL managers decided they had achieved their purpose of "bridging the gap" in introducing this new technology. CERL entered into agreements with GIS vendors, and helped installations transition their data to proprietary systems. Army research programs were directed to new challenges.

Fortunately, in the years since CERL stopped active development and support of GRASS, the Universities of Hannover (Germany), Baylor, Texas (U.S.A.), and recently the ITC-irst – Centro per la Ricerca Scientifica e Tecnologica (Italy) have continued to coordinate the development of GRASS GIS, performed by a team of developers from all over the world. Thanks to their efforts, GRASS GIS keeps getting better, and valuable and reliable Open Source GIS capabilities are still available through the Internet.

Those of us at CERL are grateful for these academic efforts. GRASS remains an unique capability that continues to play an important role in education and in the advancement of scientific understanding and resource management. The analysis tools within GRASS and the access to source code provide important benefits in our ability to understand and model geospatial phenomena. Plus, developers of this Open Source GIS continue to pioneer and advance capabilities that later emerge in the proprietary geospatial marketplace.

Thanks to the authors, this book should help sustain these important roles for GRASS GIS for years to come.

Preface to the First Edition

Geographical Resources Analysis Support System (GRASS) is the largest Free Software Geographical Information System (GIS) project and by the size of the code it belongs to the top ten list of all Open Source projects (`http://www.codecatalog.com`). The release of GRASS 5.0beta under GNU General Public License (GPL) in October 1999 protects the software authors from misuse of their developments, while offering full insight into the system. Users can analyze the internally used methods, understand their functionality, modify the programs to meet their needs, and correct or update the modules.

GRASS was developed in 1982 - 1995 by the U.S. Army Corps of Engineers Construction Engineering Research Laboratory (CERL) in Champaign, Illinois to support land management at military installations. During the late eighties, CERL published GRASS with its complete source code on the Internet. Expansion of the Internet helped to establish GRASS worldwide. In 1995, CERL withdrew from further GRASS development. In 1997, GRASS 4.2 was published by the Baylor University. The GRASS 4.2.1 release, published in 1998, was coordinated by this book's author at the Institute of Physical Geography and Landscape Ecology, University of Hannover. Nearly all known software errors were removed and about 50 new modules were added. The development of the GRASS 5.0 release started in 1999. Since 2001, the "GRASS Development Team" has its headquarters at ITC-irst (Centro per la Ricerca Scientifica e Tecnologica), Trento, Italy.

GRASS 5.0.0 was officially released in 2002 with substantial improvements over GRASS 4.x, including floating point raster data support and interactive 3D visualization. At the same time, the development work on GRASS 5.7 has started by implementing the 3D multilayer vectors, 3D TINs, multiple vector attributes with database support, and an update of 3D raster data format, paving the path for GRASS 6 to become a 3D/4D GIS. The increased activity in GRASS development in the year 2002 was accompanied by the First Inter-

national Open Source / Free Software GIS - GRASS users conference held in September 2002 in Trento, Italy, and by the publication of the first edition of this book.

The book has its own history. It started as "GRASS Recipes" written in 1995 for students at the Institute of Landscape Architecture, University of Hannover. In 1996, the first continuous German text was written and after several updates, it was finally published in "Geosynthesis" series at the Geographical Institute, University of Hannover. The first english edition of the book, published in June 2002, was the result of collaborative work of a number of translators and a new coauthor. It was substantially rewritten including updates reflecting the improvements developed for the GRASS 5.0 release. Several extended chapters were added to introduce more advanced topics and to explain the use of GRASS together with other Open Source software tools. The first edition was written for the GRASS 5.0pre3 release.

The second edition is based on the GRASS 5.3 release and includes numerous updates reflecting the enhancements of the system and the feedback that we have received from our readers. Because GRASS is updated fairly frequently there may be some differences between the command options and parameters in this book and the latest release. It is therefore useful to verify the most recent command usage in the related manual page.

This book was written for experienced GIS users who want to learn GRASS, as well as for the Open Source software users who are GIS newcomers. Therefore, an introduction to UNIX/Linux and a general chapter on GIS are preceding the GRASS chapters. Then the raster, vector, site, satellite and aerial imagery processing is described followed by notes on programming and application examples, including the work with other Open Source tools. An appendix provides an overview of GRASS modules, as well as additional technical information. A wide range of examples illustrating GRASS applications for spatial analysis, modeling and visualization are provided as an inspiration for the readers own GIS analysis.

The GRASS project's Web site, providing access to the GRASS software and documentation, can be reached at "GRASS European Headquarters" at `http://grass.itc.it` and a number of mirror sites including the "GRASS U.S.A. mirror" at `http://grass.baylor.edu`.

MARKUS NETELER, HELENA MITASOVA

Preface to the Second Edition

Since the first edition of this book was published in 2002, GRASS has undergone major improvements. The second edition includes numerous updates related to the new development and its text is based on the GRASS 5.3 version from December 2003. Besides changes related to GRASS enhancements, the introductory chapters have been re-organized. The UNIX/Linux section was omitted because most readers are already familiar with the system. The second chapter now describes basic GIS concepts and coordinate systems. Experienced GIS users may skip this chapter and start working with GRASS in chapter three. The properties of GRASS raster, vector, and site data are presented in chapter four, which also includes extensive material on importing external data in various formats. The following chapters of the book have structure similar to the first edition, with changes related to GRASS updates and improvements in technical accuracy and clarity. Most of these improvements were based on valuable feedback from readers. The sample data set used throughout the book has been updated with new data and is now available as spearfish_grass53data.tar.gz on the GRASS Web site. We hope that the book will not only help users to learn GRASS, but that it will also inspire the development of new and original GIS methods and tools.

MARKUS NETELER, HELENA MITASOVA

Acknowledgments

First and foremost, we would like to thank the large number of developers who designed, implemented, and enhanced GRASS over the 20 years of its existence. We especially appreciate the help from the members of the current GRASS Development Team who answered our numerous questions and implemented the bug fixes and improvements that we needed to make this book better.

We would like to acknowledge the contributions of Jaro Hofierka from the University of Presov, Slovakia, and Roger Bivand, Norwegian School of Economics and Business Administration, who helped with several chapters. We are grateful to Aldo Clerici, Parma University, for his excellent technical comments and to Andy Mitas, North Carolina State University (NCSU) for his valuable editorial review. Our thanks go to the translators of the original German publication whose volunteer contribution was very helpful for writing this book in a relatively short time. The reviews and comments by Lorenzo Potrich, Stefano Menegon, Stefano Merler (ITC-irst), Otto Dassau (GDF Hannover bR), Manfred Redslob (University of Hannover), Marcel Suri (Geomodel s.r.o.), and by Lubos Mitas were helpful for improving the technical accuracy and clarity of the book. We also thank Michael M. Kimberley and Thomas Drake (NCSU) for reviews of the key chapters. Robert Austin and David Pierson (NCSU) provided important technical assistance.

We greatly appreciate the support of our research work related to this book by Cesare Furlanello and ITC-irst – Centro per la Ricerca Scientifica e Tecnologica, Italy, as well as by Russell Harmon from the Army Research Office and by the National Research Council.

Previous support for the GRASS software development by William D. Goran and USA CERL, Douglas Johnston and the University of Illinois Geographic Modeling Systems Laboratory, as well as the University of Hannover, Institute of Physical Geography, is also acknowledged.

We are grateful to James Westervelt, Michael Shapiro, David Gerdes and William Brown for major code design of GRASS and, with assistance of many more developers at USA CERL, the coding of the GRASS 4.x series as well as most of the core GRASS 5.0 implementations.

Chapter 1

OPEN SOURCE SOFTWARE AND GIS

Over the past decade Geographical Information Systems (GIS) have evolved from a highly specialized niche to a technology that affects nearly every aspect of our lives, from finding driving directions to managing natural disasters. While just a few years ago the use of GIS was restricted to a group of researchers, planners and government workers, now almost everybody can create customized maps or overlay GIS data. On the other hand, many complex problems related to urban and regional planning, environmental protection, or business management, require sophisticated tools and special expertise. Therefore the current GIS technology spans a wide range of applications from viewing maps and images to spatial analysis, modeling and simulations.

GIS is often described as integration of data, hardware, and software designed for management, processing, analysis and visualization of georeferenced data. Its software component has a profound impact on the capabilities to effectively use the spatial data for solving a wide range of problems. To ensure the continuous innovation and improvement of the GIS software, existence of diverse approaches to GIS software development is crucial. Besides the widely used proprietary systems, an Open Source GIS plays an important role in adaptation of GIS technology by stimulating new experimental approaches and by providing access to GIS for the users who cannot or do not want to use proprietary products.

1.1. OPEN SOURCE CONCEPT

The idea of Open Source software has been around for almost as long as software has been developed. The results of research and development at the universities and government laboratories have been often made available in the

form of Public Domain software packages. Richard M. Stallman first defined the concept of Free Software in form of four freedoms:

0. freedom: The freedom to run the program, for any purpose.
1. freedom: The freedom to study how the program works, and adapt it to your needs.
2. freedom: The freedom to redistribute copies.
3. freedom: The freedom to improve the program, and release your improvements to the public, so that the whole community benefits.

Software following these four principles is called "Free Software". In 1984, Richard M. Stallman started to work on the GNU-Project and in 1985 he created the "Free Software Foundation" to support the Free Software concept. The license of the GNU-Project, the GNU General Public License not only grants the four freedoms described above, but it also protects them. Because of this protection the GPL is the most widely used license for Free Software nowadays. You can learn more about the ideas behind the Open Source at the Open Source[1] and Free Software[2] Web sites.

The basic idea is based on the assumption that by allowing the programmers to read, redistribute, and modify the source code, the software evolves. It gets improved, software errors (often called "bugs") are fixed and capabilities expanded. And, depending on the level of programmer's involvement and expertise, this can happen at a speed that may be quite impressive compared to the pace of conventional software development.

Full access to the source code is particularly important for GIS because the underlying algorithms can be complex and can greatly influence the results of spatial analysis and modeling. To fully understand system's functionality, which is not as obvious as it may be for example for a word processing software, it is important to be able to review and verify the implementation of a particular function. While an average user may not be able to trace bugs within a complex source code, there is a number of specialists willing to test, analyze and fix the code. The different backgrounds and expertise of these developers contribute to the synergethic effects leading to faster and more cost effective software development of a stable and robust product.

Over the past few years several Open Source GIS and GPS projects have been established with different goals. Most of them are listed at the "FreeGIS portal" Web site[3]. Smaller projects are usually based on individual developer's initiative, when the lack of available software for a specific application is solved by his own development and the result is then made available to the public on the Internet. Depending on the level of required expertise other programmers may join the project and further develop, improve and extend these

tools. Some projects are finished quickly, others evolve over time. In general the Open Source development is very dynamic. The Open Source licenses and the free access through the Internet enable the new contributors take over an abandoned project and continue the development. The overall idea differs significantly from the strategies used in the proprietary GIS development industries.

1.2. GRASS AS AN OPEN SOURCE GIS

GRASS (Geographical Resources Analysis Support System) is a raster/vector GIS combined with integrated image processing and data visualization subsystems. It includes more than 350 modules for management, processing, analysis and visualization of georeferenced data. As we have mentioned in the Preface the key development in the recent GRASS history was the adoption of GNU GPL (General Public License, see `http://www.gnu.org`) in 1999. By this, GRASS embraces the Open Source philosophy, well known from the GNU/Linux development model, which stimulated its wide acceptance (Raymond, 1997 and Raymond, 1999, for a discussion see also Wheeler, 2003). This license protects the GRASS contributors against misuse of their code contribution within proprietary projects which do not allow free access to their source code. The GPL ensures that all code based on GPL'ed code must be published again under GPL. The benefits of using other developers' code further increases the motivation to participate. For the GRASS users the license offers various advantages besides full access to the source code, especially the low costs, access to the new features and capabilities developed between the releases and possibility to provide releases more often than it is common for proprietary products. Finally, full access to the source code is also an investment protection for the future. In case that the project is withdrawn by the current developers, others may take over the development, while keeping free access to the source code.

Unlike most proprietary GIS, GRASS provides complete access to its internal structure and algorithms. Advanced users who want to write their own GIS modules may therefore learn from existing modules as well as by reading the "GRASS Programmer's Manual". The documented GRASS GIS libraries with the Application Programming Interface (API) make the new module development more efficient and allow to integrate new functionality into GRASS. Applications can be also written with shell scripts to automate the GIS workflow.

The GRASS Development Model is similar to other Open Source projects (Figure 1.1). The backbone of the project is the Internet which supports the software distribution, user support, centralized management of the GRASS

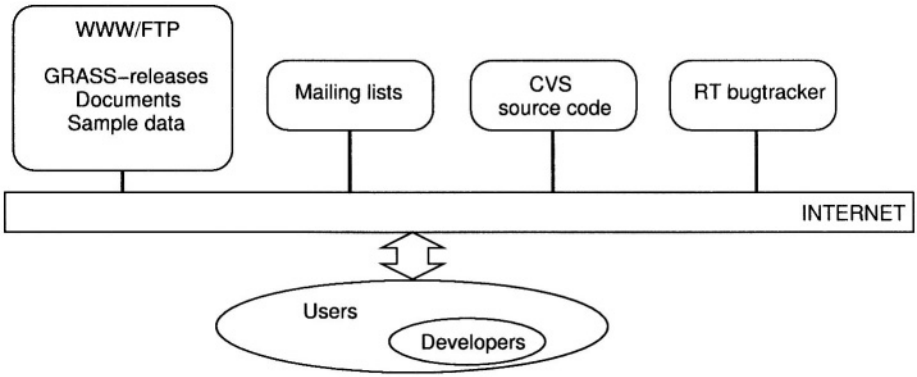


Figure 1.1. GRASS Development Model: Developers' and users' interaction with semi-automated development tools over Internet

development through CVS (Concurrent Versioning System), as well as a bug-tracking system and mailing lists. The GRASS Development Team is coordinated at ITC-irst – Centro per la Ricerca Scientifica e Tecnologica, Trento (Italy) and includes developers from all over the world. The team continues to work on improving the GRASS capabilities.

GRASS is available via Internet and on CD-ROM as precompiled binary versions for different UNIX, MacOS X and MS-Windows platforms along with the complete C-source code. While through the GPL GRASS is Free Software with protection of the authors' rights, commercial services related to GRASS can be offered and are welcome by both the developers and users community.

1.3. HOW TO READ THIS BOOK

This book focuses on the basic principles and functionality of GRASS. After a brief introduction to GIS principles, map projections and coordinate systems are explained. GRASS is introduced in the third chapter using a sample database provided on the related Web site.[4] The next chapter describes the properties of GRASS raster, vector and site data and provides extensive information on import and export of a wide range of data formats. Management, display, analysis and modeling using raster, vector, and point data is covered in the next three chapters, again using hands-on examples based on the sample database. Interactive visualization and map creation is covered at a basic level needed to communicate the results of a GIS project effectively. An extensive chapter is devoted to the satellite image processing and analysis as a special case of raster data application, followed by explanation of orthophoto creation

from scanned aerial imagery. The next chapter provides an introduction to GRASS scripting and programming. Specific applications of GRASS in the area of natural resources are illustrated within the next chapter. The last chapter demonstrates the use of GRASS with other Open Source software. The Appendix provides text file processing explanations, equations used in some of the modules and an example for WebGIS implementation. The sections for more experienced GRASS users are marked by an "advanced" arrow (⇑). References to literature provide access to detailed information about the given topic.

We use the following conventions throughout the book. Commands which you can type in are written in typewriter font, for example: `r. mapcalc`. Terminology related to GRASS is written in capital letters, such as LOCATION, MAPSET, DATABASE, and GRID RESOLUTION. Wherever [. . .] appears within the description of GRASS workflow, we have omitted some less important screen output. Long lines representing UNIX or GRASS commands are broken with \; this means that the command continues on next line. This character is usually not necessary when typing, we often used it here for formatting reasons. If you use \, be sure not to have blank space after the \ character. Otherwise the subsequent line(s) are ignored. Text from the graphical user interface menu's is written within quotes, for example: "Display".

You can download ready-to-use databases which we use throughout the book as well as updates to this book from the related Web site.[5]

NOTES

1 Open Source Web Site, `http://www.opensource.org`
2 Free Software pages, `http://www.gnu.org/philosophy/free-software-for-freedom.html`
3 FreeGIS Web Portal, `http://www.freegis.org`
4 GRASS Web site, `http://grass.itc.it/data.html`
5 Book related Tutorials Web site (book data sets, scripts etc.), `http://mpa.itc.it/grasstutor/`

Chapter 2

GIS CONCEPTS

To use GIS effectively, it is important to understand the basic GIS terminology and functionality. While each GIS software has slightly different naming conventions, there are certain principles common to all systems. At first, we briefly describe the GIS basics in general (for in depth information read Longeley et al., 2002, Clarke, 2002, or Burrough and McDonnell, 1998) and then we explain the principles of map projections and coordinate systems that are used to georeference the data.

2.1. GENERAL GIS PRINCIPLES

Data in the GIS database provide a simplified, digital representation of Earth features for a given region. Georeferenced data can be organized within GIS using different criteria, for example, as thematic layers or spatial objects. Each thematic layer can be stored using an appropriate data model depending on the source of data and their potential use.

2.1.1 Geospatial data models

Georeferenced data include a *spatial* (geometrical or graphical) component describing the location or spatial distribution of geographic phenomenon and an *attribute* component used to describe its properties. The spatial component can be represented using one of the two basic approaches (Figure 2.1):

- *field* representation, where each regularly distributed point or an area element (pixel) in the space has an assigned value (a number or no-data), leading to the *raster data model;*

- *geometrical objects* representation, where geographic features are defined as lines, points, and areas given by their coordinates, leading to the *vector data model.*

Depending on the scale, the representation of a geographic feature can change; for example, a river can be handled as a line at small scale or as a continuous 3D field (body of water) at a large scale. Similarly, a city can be represented as a point or an area. Note that we use the terms small and large scale in the cartographic sense, for example, 1:1million is small scale, 1:1000 is large scale.

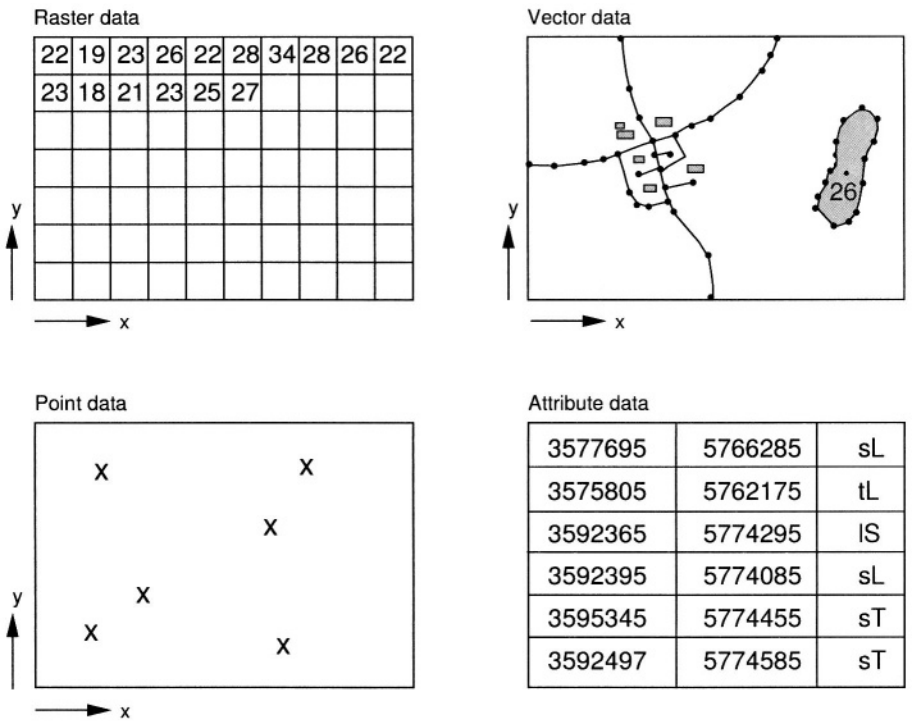


Figure 2.1. Data models in GIS: raster, vector, point data and attributes:
Raster data: rows and columns of values representing spatial phenomenon;
Vector data: representation by lines and areas;
Point data: values are assigned to individual points which are often irregularly distributed;
Attributes: descriptive data stored in a database table

To effectively use GIS, it is useful to understand the basic properties and applications of each data model (in older GIS literature, the raster and vector data models have been often referred to as raster and vector data formats).

Raster data model. Raster is a regular matrix of values (Figure 2.1). If the values are assigned to *grid points,* the raster usually represents a continuous field (elevation, temperature, chemical concentration) and is sometimes called *lattice.* If the values are assigned to *grid cells* (area units), it represents an image (satellite image, scanned map, converted vector map). If the cell values represent category numbers, one or more attributes can be assigned to that cell using a database. For example, a soil type number 3 can have attributes describing its texture, acidity, color and other properties. The grid cells are organized and accessed by *rows* and *columns.* The area represented by a square grid cell is computed from the length of its side, called *resolution.* Resolution controls the level of spatial detail captured by the raster data. Most data are represented by a 2D raster, with the grid cell (unit area) called a *pixel;* volume data can be stored as a 3D raster with a unit volume called a *voxel.* General d-dimensional raster formats are used for spatio-temporal or multispectral data (e.g. HDF format[1]).

The raster data model is often used for bio-physical subsystems of the geosphere such as elevation, temperature, water flow, or vegetation. However, it can also be used for data usually represented by lines and polygons such as roads or soil properties, especially for scanned maps. The raster data model was designed with a focus on analysis, modeling and image processing. Its main advantage is its simplicity, both in terms of data management as well as the algorithms for analysis and modeling, including map algebra. This data model is not particularly efficient for networks and other types of data heavily dependent on lines, such as property boundaries. GRASS has extensive support for the raster data model.

Vector data model. Vector data model is used to represent areas, lines and points (Figure 2.1). In this section, we describe the vector data model using GRASS terminology; however, in other systems the definitions may be slightly different.

The vector data model is based on *arc-node* representation, consisting of non-intersecting curves called *arcs.* An arc is stored as a series of points given by (x,y) or (x,y,z) coordinate pairs or triplets (with height). The two endpoints of an arc are called *nodes.* Points along a curve are called *vertices.* Two consecutive (x,y) or (x,y,z) pairs define an *arc segment.* The arcs form higher level map features: *lines* (e.g., roads or streams) or *areas* (e.g., farms or forest stands). Arcs that outline areas (polygons) are called *area edges* or *area lines.* Each map feature is assigned a category number which is used to link the geo-

metric data with descriptive, attribute data (such as category labels or multiple attributes stored in a database). For example, in a vector map layer "roads", a line can be assigned category number 2 with a text attribute "gravel road" and a numerical attribute representing its width.

In addition to the coordinate information, the vector data model often includes information about the data *topology* which describes the relative position of objects to each other. The rules which apply to the vector data with topology description are explained in the Section 6.1.1.

Linear features or polygon boundaries are drawn by straight lines connecting the points defining the arc segments. To reduce the number of points needed to store complex curves, some GIS include mathematically defined *curve sections* or *splines* which are used to compute the points with the required density at the time of drawing.

Vector data are most efficient for features which can be described by lines with simple geometry, such as roads, utility networks, property boundaries, building footprints, etc. Continuous spatial data can be represented by isolines or various types of irregular meshes using the vector data model; however, such representations usually lead to more complex algorithms for analysis and modeling. GRASS 5.3 provides basic support for the 2D vector model, while the GRASS 5.7 introduces 3D multiattribute vector model.

Point data model. The point data model is a special case of the vector data model. It is a set of independent points given by their coordinates representing point features (e.g. a city or a church) or samples of continuous fields (e.g., elevation, precipitation), often irregularly distributed. A value or a set of attributes (numerical or text) is assigned to each point. Point data are often represented using the vector data model. GRASS up to version GRASS 5.3 allows the user to store point data in a special data model designated as sites while GRASS 5.7 manages point data in the Vector model.

Attributes – GIS and databases. Attributes are descriptive data providing information associated with the geometrical data. Attributes are usually managed in external or internal GIS database management systems (DBMS). The databases use the corresponding coordinates or identification numbers to link the attribute to the geometrical data. Other systems such as PostGIS[2] also allow the user to store geometrical data into the database. GRASS 5.3 offers a limited internal database and several interfaces to external databases (PostgreSQL, ODBC interface to various DBMS). The GRASS internal database supports only a single attribute for each vector object or cell category. GRASS 5.7 provides extended capabilities as it includes a SQL-DBMS engine.

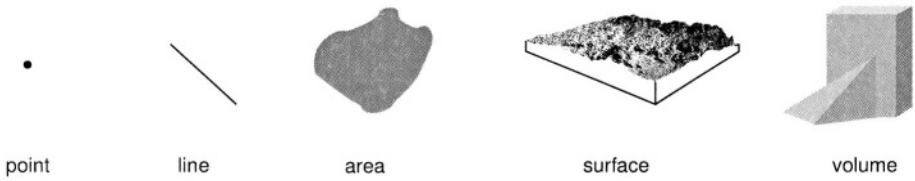


Figure 2.2. Data dimensions in a Geographical Information System (after Rase, 1998:19)

Data model transformations. The same phenomenon or feature can be represented by different data models. GIS usually includes tools for transformation between the vector, raster and site data. For example, elevation can be measured as point data, then interpolated into the raster map layer which is then used to derive contour lines as vector data. Note that transformations between different data models are usually not lossless (there can be a loss or distortion of information due to the transformation).

Dimensions of geospatial data. In general, Earth and its features are located and evolve in 3D space and time. However, for most applications a projection of geospatial data to a flat plane is sufficient; therefore two-dimensional representation of geographical features (with data georeferenced by their horizontal coordinates) is the most common. Elevation as a third dimension is usually stored as a separate data layer representing a surface within three-dimensional space (often referred to, not quite correctly, as a 2.5-dimensional representation, Figure 2.2). Elevation can be also added as a z-coordinate or an attribute to vector and point data. If there is more than a single value associated with a given horizontal location, the data represent a volume and are three-dimensional (e.g. chemical concentrations in groundwater, or air temperature). Three-dimensional data can change in time, adding the fourth dimension. GIS provides the most comprehensive support for 2D data. GRASS 5.3 includes a 3D raster model for volume data and a multidimensional, multi-attribute site data format (see Brandon et al., 1999; Neteler, 2001a); however, only a limited number of modules is available for volume data.

2.1.2 Organization of GIS data

GIS can be implemented as a comprehensive, multipurpose system (GRASS, ArcGIS), as a specialized, application oriented tool (MapQuest), or as a subsystem of a larger software package supporting handling of geospatial data needed in its applications (e.g. hydrologic modeling system, geostatistical analysis software, or a real estate services Web site). The multipurpose

systems are often built from smaller components or modules which can be used independently in application oriented systems.

The multipurpose GIS usually stores the georeferenced data as thematic *map layers*. Each geographic feature or theme, such as streams, roads, vegetation, or cities is stored in a separate map layer using the vector or raster data model. The map layers can then be combined to create different types of new maps as well as perform analysis of spatial relations. GRASS and most of the proprietary GIS products are based on this data organization.

For certain applications, especially those based on discrete, object based representation of geographical features, the *object oriented* approach to data management is used. Within this approach, data are stored as closed objects with coordinates and attribute information. Objects include characteristics and methods. The characteristics describe the data structure, the methods include the information about the data exchange with other objects. The advantage of this concept is in the possibility to generate complex data structures connected to objects, allowing an efficient management of data for such applications as utilities or land register. The main disadvantage of this concept is that the management of continuous data, which are important for physical geography and landscape ecology, is unfavorable in object oriented form. GRASS does not use the object oriented approach to data organization, although some modules (`nviz`) and add-on libraries (`GDAL`) have object oriented design.

A large volume of geospatial data is nowadays distributed through *Internet based GIS*. The data sets are stored on central server(s) and users access the data as well as the display and analysis tools through the Internet. Examples are the interactive National Atlas of the U.S.[3], MapQuest[4] or UMN/MapServer Gallery[5]. Almost every multipurpose GIS software includes tools supporting development of Web-based applications. GRASS can be used with UMN/MapServer, an Open Source project for developing Web-based GIS applications which supports a variety of spatial requests like making maps, scale-bars, and point, area and feature queries (see Chapter 13). Other projects such as ICENS Spatial Information System[6] and Grules[7] are using JAVA to connect GRASS to the Internet. Internet GIS can be enhanced by interactive 3D viewing capabilities using GeoVRML[8] as well as by multimedia features adding photographs, video, animations or sound to the georeferenced data.

2.1.3 GIS functionality

While creating digital and hardcopy maps has been the core GIS function over the past decade, the emphasis is shifting towards spatial analysis and modeling. GIS functionality is rapidly evolving and currently covers a wide range of areas, for example (read in more detail at Wadsworth and Treweek, 1999):

- integration of geospatial data from various sources: projections and coordinate transformations, format conversions, spatial interpolation, transformations between data models;

- visualization and communication of digital georeferenced data in form of digital and paper maps, animations, virtual reality (computer cartography);

- spatial analysis: spatial query, spatial overlay (combination of spatial data to find locations with given properties), neighborhood operations, geostatistics and spatial statistics;

- image processing: satellite and airborne image processing, remote sensing applications;

- network analysis and optimization;

- simulation of spatial processes: socioeconomic such as transportation, urban growth, population migration as well as physical and biological, such as water and pollutant flow, ecosystem evolution, etc.

This functionality is used to solve spatial problems in almost every area of our lives. Here are a few examples. In the area of socioeconomic applications, GIS can be used to find directions, locate a hospital within a given distance from a school, find optimal locations for a new manufacturing facility, design voter districts with given composition and number of voters, identify crime hot spots in a city, select optimal evacuation routes, manage urban growth. GIS plays an important role in conservation of natural resources and management of natural disasters, such as identification and prevention of soil erosion risk, forest resource management, ecosystem analysis and modeling, planning of conservation measures, flood prediction and management, pollutant modeling, etc. GIS is also being increasingly used in agriculture, especially in the area of precision farming.

2.2. MAP PROJECTIONS AND COORDINATE SYSTEMS

The basic property of GIS, as opposed to other types of information systems, is that the stored data are georeferenced. That means that the data have defined their location on Earth using coordinates within a georeferenced coordinate system. The fact that the Earth is an irregular, approximately spherical object

Ellipsoid name	*Region of use*
Airy 1858	Great Britain
Airy modified	Ireland
Australian National	Australia
Bessel 1841	Austria, Chile, Croatia, Czech Rep., Germany, Greece, Indonesia, Netherlands, Slovakia, Sweden, Switzerland
Bessel modified	Norway
Clarke 1880	Africa, France
Clarke 1866	North America, Philippines
Everest 1830	Afghanistan, Myanmar, India, Pakistan, Thailand, and other countries in southern Asia
GRS 1980	North America, worldwide
Hayford (International) 1909	Belgium, Finland, Italy, all countries using ED50 system
New International 1967	many other regions
Krassovsky 1938	Albania, Poland, Romania, Russia and neighboring countries
WGS 1984	North America, worldwide
WGS 1972	NASA satellite

Table 2.1. Selected standard ellipsoids as used in various countries

makes the definition of an appropriate coordinate system rather complex. The coordinate system either has to be defined on a sphere or ellipsoid, leading to a system of geographic coordinates or the sphere has to be projected on a surface that can be developed into a plane where we can define the cartesian system of coordinates (*easting, northing and elevation;* see Sections 2.2.2).

Because GRASS keeps the projects organized by LOCATIONs, where each LOCATION has a unique map projection and coordinate system, it is important to understand the relevant terminology before starting to work with geospatial data.

2.2.1 Map projection principles

When working with GRASS, the projection and coordinate system must be defined when a new project (LOCATION in GRASS terminology) is defined. The map projection definition is stored in an internal file within the given LOCATION. It is used whenever the data need to be projected into a different projection or when calculations requiring information about the Earth's curvature are performed. Different parameters are needed to define different projections and coordinate systems; therefore, it is important to understand the map projection terminology.

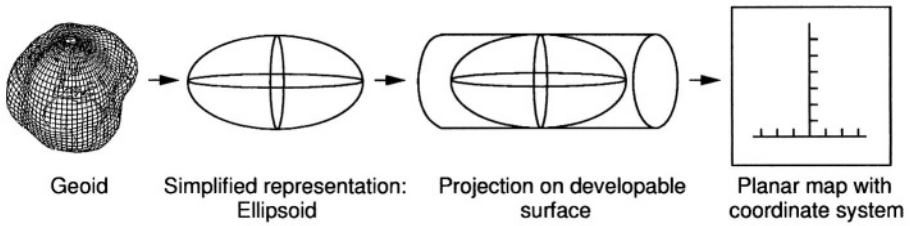


Figure 2.3. Earth's surface representation in map projections and coordinate systems

Shape of the Earth. Shape of the Earth is usually approximated by a mathematical model represented by an *ellipsoid* (also called a spheroid). A variety of cartographic ellipsoids have been designed to provide the best-fit properties for certain portions of the Earth's surface, see for example Table 2.1.

While the ellipsoid describes the shape of the Earth by a relatively simple mathematical function, the *geoid,* a physical approach to the description of the Earth's shape, undulates responding to the distribution of the Earth's mass which locally varies. The geoid is the equipotential surface of Earth's gravity field and corresponds to the mean sea level. For map projections, the ellipsoids are usually sufficient for horizontal positioning; however, the geoid has to be used for exact elevation calculations.

Geodetic or map datum. A set of constants specifying the coordinate system used for calculating the coordinates of points on Earth is called a *geodetic datum.* Horizontal datums define the origin and orientation of a coordinate system used to calculate the horizontal coordinates (usually northing and easting). Vertical datums define the coordinate system origin for calculating the elevation coordinate (mean sea level). For map layers to match, their coordinates must be computed using the same datum. Different datums mean a shift in the origin of the coordinate system, and that means a shift of the entire map.

Map projection. To transform a curved Earth surface into a plane (flat sheet of paper or a computer screen), a *map projection* is used. Direct projection of a spherical object to a plane cannot be performed without distortion. The most common approach is to project the spheroid onto a *developable surface,* such as a cylinder or a cone which can be developed into a plane without deformation (tearing or stretching), see Figure 2.3. A large number of different projections have been designed with the aim to minimize the distortion and preserve certain properties. In general, the projections can be divided into three major groups (for a mathematical description refer to Bugayevskiy and Snyder, 2000:20-22):

- *conformal,* preserves angles (shapes for small areas), used for navigation and most national grid systems;
- *equidistant,* preserves certain relative distances, used for measurement of length;
- *equivalent,* preserves area, used for measurement of areas.

Each of these properties (angle, distance, area) is preserved at the expense of the others. Because there is no perfect solution, the map projection is selected depending on the application. Most coordinate systems used for land surface mapping use conformal projections.

The developable surfaces can either touch the spheroid (tangent case) or intersect it (secant case). Based on the geometry of the developable surface, the projections can be divided into:

- *cylindrical,* which transform the spherical surface to a tangent or secant cylinder;
- *conic,* which use the tangent or secant cone;
- *azimuthal,* which use a tangent or secant plane (flat sheet).

The points or lines where a developable surface touches or intersects the spheroid are called *standard points* and *standard lines* with zero distortion (e.g. standard parallel for tangent cone or two standard parallels for secant cone). That means that the projected maps (or in the GIS the projected data) do not have uniform scale for the entire area, and the true map scale is preserved only along the standard lines. To minimize distortions, some projections reduce the scale along the standard parallel(s) or central meridian(s). This is expressed as a *scale factor* smaller than 1.0 in the definition of such a projection.

Transverse projections use developable surfaces rotated by 90° so that the standard (tangent) line is a meridian called *central meridian* instead of a standard parallel. *Oblique projections* may use any rotation defined by azimuth where *azimuth* is an angle between a map's central line of projection and the meridian it intersects, measured clockwise from north. Snyder, 1987, provides an excellent manual on map projections with map examples for many important projections.

Coordinate system. To accurately identify a location on Earth, a *coordinate system* is required. It is defined by its origin (e.g. prime meridian, datum), coordinate axes (e.g. x, y, z), and units (angle: degree, gon, radiant; length: meter, feet).

Projection Type	*Country*
Transverse Mercator	Albania, Australia, Austria, Denmark, Finland, Germany, Great Britain, Ireland, Italy, Luxembourg, Norway, Poland, Portugal, Russia, Spain, Sweden, USA
Oblique Mercator	Hungary, Madagascar, Malaysia, Switzerland
Lambert Conformal Conic	Belgium, France, Portugal, USA
Stereographic	Netherlands (oblique aspect), Poland, Romania, UPS (polar regions)

Table 2.2. Selected projections used in various countries

The following general coordinate systems are commonly used in GIS:

- *geographic* (global) coordinate system (latitude-longitude);
- *planar (cartesian) georeferenced* coordinate system (easting, northing, elevation) which includes projection from an ellipsoid to a plane, with origin and axes tied to the Earth surface;
- *planar non-georeferenced* coordinate system, such as image coordinate system with origin and axes defined arbitrarily (e.g. image corner) without defining its position on the Earth.

Note that for planar georeferenced systems *false easting* and *false northing* may be used. These are selected offset constants added to coordinates to ensure that all values in the given area are positive.

For mapping purposes, each country has one or more *national grid systems.* Information about national grid systems can be obtained from the national cartographic institutes or from the Internet ASPRS site[9]. A national grid system is defined by a set of parameters such as ellipsoid, datum, projection, coordinate system origin and axes, etc. Examples of worldwide and national grid systems are UTM (Universal Transverse Mercator), Gauss-Krüger, Gauss-Boaga, or State Plane, with the projections listed in the Table 2.2. Information about the grid system used to georeference digital geospatial data is a crucial component of the metadata and allows the user to integrate and combine data obtained from different sources.

2.2.2 Common coordinate systems

Geographic coordinate system: latitude-longitude. The most common coordinate system used for the global data is the spherical coordinate system which determines the location of a point on the globe using latitude and longitude. It is based on a grid of meridians and parallels, where *meridians* are the

longitude lines connecting the north and south poles and *parallels* are the latitude lines which form circles around the Earth parallel with the *equator.* The longitude of a point is then defined as an angle between its meridian and the *prime meridian* (0° east, passing through the Royal Observatory in Greenwich, near London, UK). The latitude of a point is defined as an angle between the normal to the spheroid passing through the given point and the equator plane. The longitude is measured 0-180° east from prime meridian and 0-180° west, where 180° longitude is the international date line. Latitude is measured 0-90° north and 0-90° south from equator.

Geographic coordinates can be expressed in two notations:

- decimal degree;
- sexagesimal degree.

Decimal values of W and S are expressed as negative numbers, N and E as positive numbers (e.g. Murcia, Spain: -1.167°, 38.0°). Values given in sexagesimal system always use positive numbers together with N, S, E, W (Murcia, Spain: 1:10:00W, 38:00:00N). It is not difficult to convert between these notations.

Universal Transverse Mercator Grid System. The Universal Transverse Mercator (UTM) Grid System is used by many national mapping agencies for topographic and thematic mapping, georeferencing of satellite imagery and in numerous geographical data servers. It applies to almost the entire globe (area between 84° N and 80° S). The pole areas are covered by the Universal Polar Stereographic (UPS) Grid System not explained here; please refer to Robinson et al., 1995 or other authors.

UTM is based on a Transverse Mercator (conformal, cylindrical) projection with strips (zones) running north-south rather than east-west as in the standard Mercator projection. UTM divides the globe into 60 zones with a width of 6° longitude, numbered 1 to 60, starting at 180° longitude (west). Each of these zones will then form the basis of a separate map projection to avoid unacceptable distortions and scale variations. Each zone is further divided into strips of 8° latitude with letters assigned to from C to X northwards, omitting the letters I and O, beginning at 80° south (Robinson et al., 1995:101).

The origin of each zone (central meridian) is assigned an easting of 500,000 m (false easting, Maling, 1992:358). For the northern hemisphere the equator has northing set to zero, while for the southern hemisphere it has northing 10,000,000 m (false northing). To minimize the distortion in each zone, the scale along the central meridian is 0.9996, leading to a secant case of the Transverse Mercator projection with two parallel lines of zero distortion. Note that UTM is used with different ellipsoids, depending on the country and time of mapping.

For GIS applications, it is important to realize that each UTM zone is a different projection using a different system of coordinates. Combining maps from different UTM zones into a single map using only one UTM zone (which can be done relatively easily using GIS map projection modules) will result in significant distortion in the location, distances and shapes of the objects that originated in a different zone map and are outside the area of the given zone. To overcome the problem, a different coordinate system should be used and the data re-projected. For a quick reference, you can find the UTM zone numbers in the Unit 013 "Coordinate System Overview" of the NCGIA Core Curriculum in GIS.[10]

Lambert Conformal Conic Projection based systems. The Lambert Conformal Conic (LCC) projection is one of the best and most common for middle latitudes. It uses a single secant cone, cutting the Earth along two standard parallels. The tangent cone version with a single standard parallel case is also used. When working with LCC based coordinate systems, the following parameters have to be provided: the standard parallel(s) (one or two), the longitude of the central meridian, the latitude of projection origin (central parallel), false easting and, sometimes, false northing (you may recall that false easting and northing are shifts of the origin of the coordinate system from the central meridian and parallel).

State Plane Coordinate System. The State Plane Coordinate System used by state mapping agencies in the U.S.A. is based on different projections depending on the individual state shape and location, usually LCC or a Transverse Mercator with parameters optimized for each state. Various combinations of datums (NAD27, NAD83) and units (feet, meters) have been used, so it is important to obtain all relevant coordinate system information (usually stored in the metadata file) when working with the data georeferenced in the State Plane Coordinate System. GIS projection modules often allow to define the State Plane system by providing the name of the state and the county, however, the parameters should always be checked, especially when working with older data.

Gauss-Krüger Grid System. The Gauss-Krüger Grid System is used in several European and other countries. It is based on the Transverse Mercator Projection and the Bessel ellipsoid. The zones are 3° wide, leading to 120 strips. The zone number is divided by 3 according to longitude of central meridian. Adjacent zones have a small overlapping area. The scale along the central meridian (scale factor) is 1.0.

Figure 2.4 illustrates the coordinate system, the x-axis is defined by the central meridian, the y-axis by the equator. The northing values are positive north

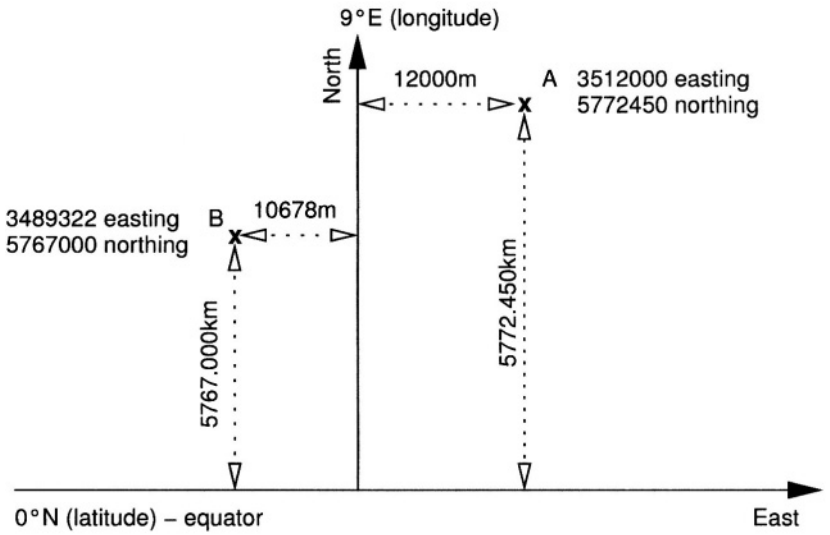


Figure 2.4. Example for the Gauss-Krüger Grid System with two points A and B

from the equator, the easting values are measured from the central meridian. To avoid negative values, a false easting of 500,000 m is defined in addition to the third of the longitude of the central meridian. For example the false easting for the 9° E central meridian is 3,500,000 m (9/3 = 3, value composed with 500,000 m to 3,500,000 m).

2.2.3 North American and European Datums

In general, a large number of georeferencing datums exists, here we focus on three examples. The North American Datum 1983 (NAD83) is a geodetic reference system which uses as its origin the Earth's center of mass, whereas the old North American Datum 1927 (NAD27) had a different origin, making it useful only in North America. GPS receivers which are mostly based on the WGS84 datum (other local datums can be selected in the GPS receiver's menu) also use the Earth's center of mass as their system's origin.

When using maps based on different datums, a datum transformation to a common datum is required. For example, a change from NAD27 to NAD83 system leads to a shift for the entire map. Overlapping maps based on different datums of the same region would not co-register properly without datum transformation. In the continental United States a few common assignments between datums and ellipsoids are in use: NAD27 datum with Clarke 1866 el-

lipsoid, NAD83 datum with GRS80 ellipsoid, and WGS84 datum with WGS84 ellipsoid.

It is important to know that the NAD27 and NAD83 datums are 2D horizontal datums used for horizontal coordinates (easting and northing). WGS84 is a 3D datum (x, y and height). Separate vertical datums used with these systems are NGVD29 and NAVD88. GRASS does not handle such separate vertical datums so these transformation needs to be done outside GRASS.

NOTES

1 HDF format and tools,
`http://hdf.ncsa.uiuc.edu`

2 PostGIS DBMS, `http://postgis.refractions.net`

3 National Atlas of the U.S., `http://nationalatlas.gov`

4 MapQuest, `http://www.mapquest.com`

5 UMN/MapServer Gallery, `http://mapserver.gis.umn.edu`

6 ICENS Spatial Information System,
`http://196.3.4.220:8000/jdb/icens.sivs?class=gis`

7 Grules (GRASS JAVA Server),
`http://grules.sourceforge.net`

8 GeoVRML, `http://www.geovrml.org`

9 Information about national grid systems: ASPRS: Grids & Datums,
`http://www.asprs.org/asprs/resources/grids/`
European coordinate systems,
`http://www.mapref.org`
A comprehensive, general list of projection transformations is available at
`http://www.remotesensing.org/geotiff/proj_list/`

10 Unit 013 Coordinate System Overview in the NCGIA Core Curriculum in GIS,
`http://www.ncgia.ucsb.edu/education/curricula/giscc/units/u013/u013.html`

Chapter 3

GETTING STARTED WITH GRASS

In this chapter we begin working with GRASS. First, we explain GRASS software installation and the structure of its database. Then we use a sample database to perform basic GIS tasks. We also include a number of examples illustrating how to start a GRASS project using different coordinate systems.

3.1. FIRST STEPS

GRASS, as a multipurpose GIS, with data organized as raster, vector and site map layers, provides a wide range of tools to support most of the GIS functionality outlined in the previous sections. The overview is given in Table 3.1. Detailed explanation of each module, often with a usage example, is given in the GRASS users manual (see your GRASS installation or Web site[1]; this manual is based on a publication by the U.S. Army CERL, 1993).

While the support for temporal and volume data in GRASS 5.3 is still limited, the GRASS 5.7 that is currently under development, is being designed as full 3D GIS with support for 3D raster, 3D vector and 3D site data (see Blazek et al., 2002; Neteler, 2001a).

3.1.1 Download and install GRASS

GRASS software can be downloaded free of charge from the main GRASS Web site:

`http://grass.itc.it`

The ITC-irst GRASS site is mirrored in several countries for faster access including the GRASS U.S.A. mirror at `http://grass.ibiblio.org`.

functionality class	*functionality*
integration of geospatial data	import and export of data in various formats
	coordinate systems transformations and projections
	transformations between raster, vector and point data models
	spatial interpolation
raster data processing	comprehensive map algebra
	surface, topographic and watershed analysis
	correlation, covariant analysis
	cost surfaces, shortest path, buffers
	line of sight, insolation
	landscape ecology measures
	expert system (Bayes logic)
vector data processing	digitizing
	overlay
	spatial autocorrelation
site data processing	multidimensional, multi-attribute site data model
	summary statistics
	site buffers
	multivariate spatial interpolation and surface analysis
	voronoi polygons, triangulation
image processing	processing and analysis of multispectral satellite data
	image rectification and orthophoto generation
	principal and canonical component analysis
	reclassification and edge detection
	radiometric correction
visualization	2D display of raster, vector and site data with zoom and pan
	3D visualization of multiple surfaces with vector and site data
	2D and 3D animations
	hardcopy postscript maps
modeling and simulations	hydrologic, erosion and pollutant transport, fire
temporal data support	time stamp for raster, vector and site data
volume data processing	3D map algebra
	volume interpolation and analysis
	volume visualization (isosurfaces)
links to Open Source tools	R-stats, gstat, PostgreSQL, UMN/MapServer, Vis5D
	GPS tools, GDAL

Table 3.1. GRASS GIS functionality

There, you can find the source code (portable version for all platforms) as well as the latest ready-to-install binaries for Linux, SUN, SGI, MacOS X and MS-Windows NT/2000/XP (requiring the Cygwin tools). For some GNU/Linux distributions, RPM packages are provided.

GRASS is also available on CD-ROM, from the FreeGIS project Web site[2] and for MacOS X from OpenOSX Web site[3]. There is a fee for packaging the CD-ROM and for the customized installation software.

On the Web site you will also find the "GDP – GRASS Documentation Project", which makes it easier to find documentation, especially for externally developed GRASS-modules and various articles. These pages can be reached at:

`http://grass.itc.it/gdp/`

Support for developers and users is provided by several mailing lists to which you can subscribe using a Web interface (see the relevant links under "Support section" at the GRASS sites). Besides the English language international mailing lists, there are also localized lists currently in Czech, German, Italian, Japanese and Polish.

GRASS binary installation. The GRASS binaries are available for several platforms. You have to download the install script `grass5install.sh` and the GRASS package `grass5package_bin.tar.gz` (the name depends on the platform). The installation itself should be done as user "root". It requires only one step (check online for appropriate file names):

```
sh grass5install.sh grass5package_bin.tar.gz
```

After successful installation, the package `grass5package_bin.tar.gz` may be deleted.

Refer to the Chapter 11 for information on the GRASS source code, the CVS server and code compilation.

3.1.2 Database and command structure

GRASS data are stored in a UNIX directory referred to as DATABASE (also called "GISDBASE"). This directory has to be created with `mkdir` or a file-manager, before starting to work with GRASS. Within this DATABASE, the projects are organized by project areas stored in subdirectories called LOCATIONS (Figure 3.1).

A LOCATION is defined by its coordinate system, map projection and geographical boundaries. The subdirectories and files defining a LOCATION are created automatically when GRASS is started the first time with a new LOCATION (see Section 3.2 for more details). Each LOCATION can have several MAPSETs (Figure 3.1). One motivation to maintain different mapsets is to store maps related to project issues or subregions. Another motivation is to support simultaneous access of several users to the map layers stored within the same LOCATION, i.e. teams working on the same project. For teams a centralized GRASS DATABASE would be defined in a network file system

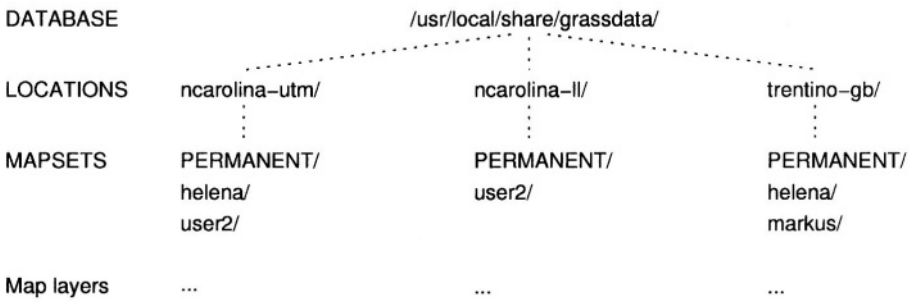


Figure 3.1. Organization of GRASS DATABASE, LOCATIONS and MAPSETs

(e.g. NFS). Besides access to his own MAPSET, each user can also read map layers in other users' MAPSETs, but he can modify or remove only the map layers in his own MAPSET.

When creating a new LOCATION, GRASS automatically creates a special MAPSET called PERMANENT where the core data for the project can be stored. Data in the PERMANENT MAPSET can only be added, modified or removed by the owner of the PERMANENT MAPSET; however, they can be accessed, analyzed, and copied into their own MAPSET by the users. The PERMANENT MAPSET is useful for providing general spatial data such as elevation model write-protected to other users who are working in the same LOCATION. To import data into PERMANENT, just start GRASS with the relevant LOCATION and the PERMANENT MAPSET. This mapset also contains the DEFAULT_WIND file which holds the default region boundary coordinate values. In all mapsets additionally a WIND file is kept for storing the current boundary coordinate values and the currently selected raster resolution.

The internal organization and management of LOCATION, MAPSETs and map layers should be left to GRASS. Operations such as renaming or copying map layers involve several internal files and should always be done through GRASS commands (we discuss this in detail in Section 3.1.4). Non-GRASS interventions are acceptable only in exceptional situations and when one has a good understanding of GRASS internal structure.

GRASS modules are organized by name, based on their function class (display, general, imagery, raster, vector or site, etc.). The first letter refers to the function class, followed by a dot and one or two other words, again separated by dots, describing the specific task performed by the module. Table 3.2 lists the most important function classes.

The general syntax of a GRASS command which is called to run a module is similar to the UNIX commands:

```
module [-flag1[flag2...]] parameter1=map1[,map2,...]\
                                  [parameter2=number...]
```

where `module` is the name of the command (see Table 3.2, optional flags enable specific features and parameters are names of input or output files or may be a constant or name of a method, symbol etc. Note that there must be no space when listing comma-separated names.

For example, `v.in.shape` is a vector command for importing ESRI SHAPE files, `r.buffer` calculates a buffer zone along raster lines and around raster areas, `d.rast` displays a raster map layer, `i.ortho.photo` creates an orthophoto from a scanned aerial image. To learn the usage of a command (syntax, flags and parameters) run the command with the `help` option, for example:

```
d.rast help
```

and to read the module-related manual pages, run (example for `d.rast`):

```
g.manual d.rast
```

To make it easier to move around the manual pages you can change the standard manual page browser temporarily to a text browser `lynx`, `netscape`, `gnome-help-browser` or `konqueror`, provided you have them installed. For example, to view a manual page under `konqueror` run (using bash shell syntax):

```
export GRASS_TEXT_BROWSER=konqueror
g.manual d.rast
```

You can also find the HTML version of the manual pages and tutorials at the GRASS Web site.

prefix	*function class*	*type of command*
d.*	display	graphical output
s.*	sites	site data processing
r.*	raster	raster data processing
i.*	imagery	image processing
v.*	vector	vector data processing
g.*	general	general file operations
m.*	misc	miscellaneous commands
p.*	paint	map creation in PPM format
ps.*	postscript	map creation in Postscript format

Table 3.2. GRASS module function classes

3.1.3 Starting GRASS with demo database Spearfish

For the following sample session we assume that you have working knowledge of starting commands, creating directories etc. You need to install the Spearfish demo data set which you can download from the GRASS Web site or get on CD-ROM.[4] It is a comprehensive set of raster, vector and site data covering two 1:24,000 scale topographic map sheets in the western part of South Dakota (SD), USA. The names of the quadrangles are Spearfish and Deadwood North, SD. The data set includes a large portion of the Black Hills National Forest (Mount Rushmore). The coordinate system is UTM (zone 13N, Transverse Mercator projection, Clarke66 ellipsoid, NAD27 datum, metric units, boundary coordinates 4928000N, 4914000S, 590000W and 609000E) with resolutions of raster map layers ranging from 20 m to 100 m. Data have been provided by the EROS Data Center, U.S. Army CERL, USGS, U.S. Census Bureau, and SPOT Image Corporation. A more complete data description can be found at the GRASS Web site.[5]

To start, you need to create your DATABASE directory, for example, under `/usr/local/share/`. Depending on your system set-up you may have to do it as a user "root" (or ask your administrator to do it) and change the permissions so that you can read, write and execute within this directory. So as "root" you run:

```
mkdir -p /usr/local/share/grassdata
chown yourname /usr/local/share/grassdata
chgrp yourgroup /usr/local/share/grassdata
chmod ug+rwx /usr/local/share/grassdata
```

With the flag `-p`, `mkdir` first creates all the non-existing parent directories, then the desired subdirectory. Then we set the permissions to "read", "write" and "execute" for the user and the group (use your selected names for `yourname` and `yourgroup` in the example above).

Then move the downloaded file `spearfish_grass53data.tar.gz` into this directory, change into it and unpack the file:

```
mv spearfish_grass53data.tar.gz /usr/local/share/grassdata
cd /usr/local/share/grassdata
tar xvfz spearfish_grass53data.tar.gz
```

The resulting list of files shows that the data are extracted into a new subdirectory `spearfish`, which is the name of your LOCATION. After unpacking, the downloaded "tar.gz" package file can be deleted.

Starting GRASS. You can now call GRASS:

```
grass53
```

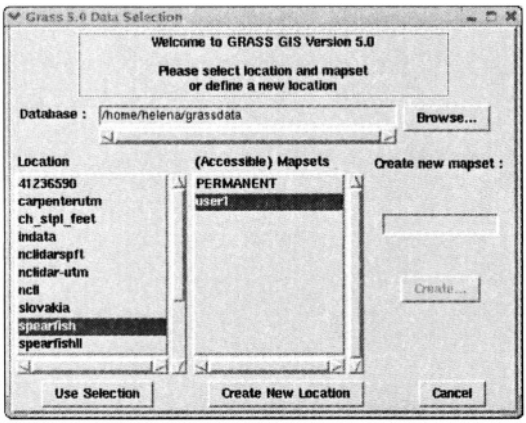


Figure 3.2. Graphical startup of GRASS

and you will see, after a welcome message, the menu for selecting a LOCATION and a MAPSET in your GRASS DATABASE (Figure 3.2). Your current directory `/usr/local/share/grassdata` will be automatically entered as a DATABASE. For LOCATION select `spearfish`; for MAPSET select `user1`. Then click on "Use selection", and you will see the license message and the command line prompt:

```
GRASS:~>
```

Now you are in GRASS and you can call GRASS modules as well as UNIX programs. You can also use a TclTk interface which opens in the upper part of your screen (Figure 3.3). If it is not already there, just type:

```
tcltkgrass &
```

Most of the GRASS commands are integrated within this interface and you can find a command for a specific task using the function menus. The interface includes a brief description of the parameters and it also displays the command line version of the module.

To list the available vector, raster and site data layers, type:

```
g.list rast
g.list vect
g.list sites
```

or, in TclTkGRASS, select under `MAP` ⇝ `LIST` ⇝ `RASTER` ⇝ `Run`. You can learn more about each data layer in terms of its minimum and maximum coordinates, resolution, and number of classes using the `*.info` commands, for example:

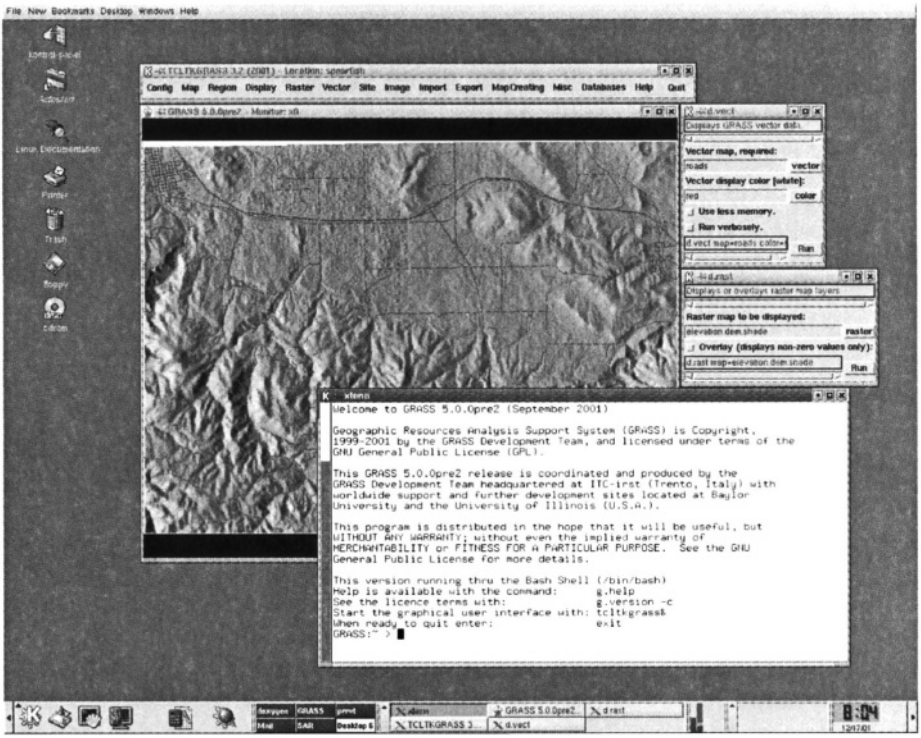


Figure 3.3. GRASS used in the KDE environment on GNU/Linux

```
r.info soils
v.info streams
s.info archsites
```

Now open a GRASS monitor so that you can display the map layers:

```
d.mon x0
```

or, in TclTkGRASS, select `DISPLAY` ⇝ `MONITOR` ⇝ `START` ⇝ `X0`. Note that there is a maximum of seven graphical monitors that can be opened at the same time. They are numbered `x0` to `x6`. The default size of the graphics monitor is relatively small so you may want to resize it to a bigger window using the mouse.

To view the raster soil map layer together with vector streams (drawn as blue lines) and archaeological sites (drawn as white squares) type (Figure 3.4):

```
d.rast soils
d.vect streams col=blue
d.sites archsites col=white type=box
```

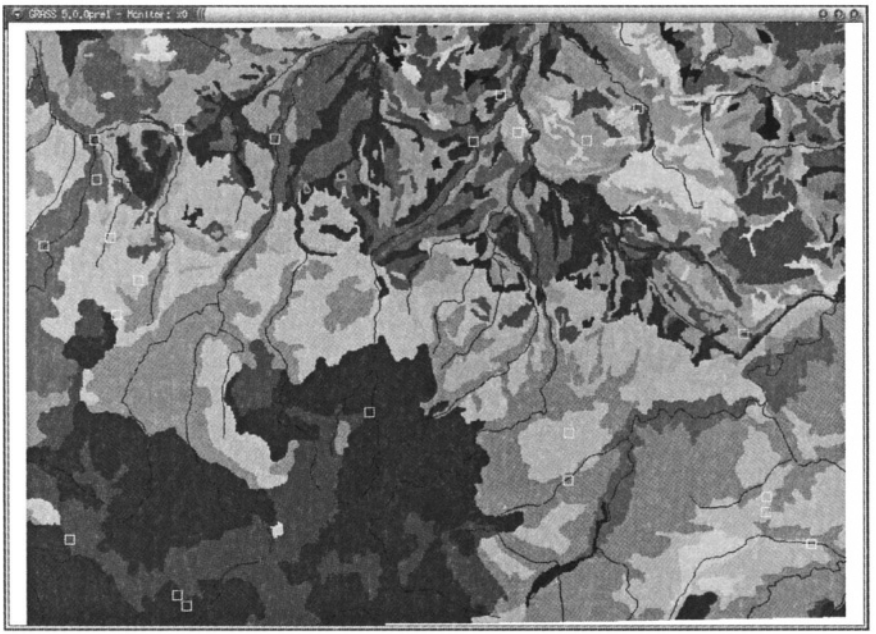


Figure 3.4. Spearfish soil raster map with overlayed vector streams and archeological sites

In the following chapters, you will see numerous examples of geospatial data processing and analysis performed with this data set; therefore, at this point, we will just show how to properly end the GRASS session. Exit `tcltkgrass` by clicking on "Quit". If there are still open monitors, close them using the mouse, then exit GRASS by typing:

```
exit
```

If you exit GRASS and you forgot to close the GRASS monitor or `tcltkgrass`, you can do it any time later by closing the relevant windows using the mouse.

3.1.4 GRASS file and location management

When working with GRASS, it is important to understand that a map layer (except the sites) is represented by several files which include the data, categories, header, and other information. To simplify procedures such as listing, copying, renaming and deleting map layers, a set of file management tools is available. These commands must be used to maintain the consistency in the GRASS DATABASE. It is not recommended to directly modify the files in the

LOCATION or MAPSET directories, unless one is experienced with the system. Note that the management modules are also applicable to other GRASS related files such as region definitions and imagery groups.

An important note for the GRASS filename convention: It is very important to avoid spaces and special characters, such as a comma, dash, exclamation mark etc. in GRASS map names. It is also useful to include at least one letter in the map name to avoid confusion with numbers being treated as values (especially when using `r.mapcalc`). While it is possible to use all combinations of characters if the map name or expression is enclosed within quotes, but it is generally safer to follow the name suggestions mentioned above.

We have already shown that we can use the command `g.list` to list available raster, vector and site map layers. To display the map layers with their titles, use `-f` flag. If you have many MAPSETs and you want to see the map layers stored only in a selected one, use the `mapset` parameter, for example:

```
g.list -f vect
g.list -f vect mapset=PERMANENT
```

Remember, when a list exceeds the terminal screen, continue with <SPACE>, go back with <b> and leave with <q>. In case you have many map layers available, you may want to list only their subset. You can use wildcards to invoke automated character or name replacement or, optionally, regular expressions. In our example we want to see all vector maps with the names starting with "r":

```
g.mlist type=vect pattern="r*"
```

To create a full copy of a map layer, use the `g.copy` module. You have to specify the map type and add an old and a new map name, separated by comma (no spaces are allowed between the names). As an example we can copy the map `railroads` from the PERMANENT MAPSET into your own MAPSET:

```
g.copy vect=railroads@PERMANENT,myrailroads
```

To rename a map, you can use `g.rename` and list the old name and the new name, separated by comma:

```
g.rename vect=myrailroads,railnetwork
```

Map removal also has to be done with a GRASS command. For example, to remove one of the recently created map copies, type:

```
g.remove vect=railnetwork
```

Multiple maps can be removed by listing them separated by comma. If you need to delete a series of maps, you may carefully (!) use the `g.mremove` module. It allows the use of wildcards similar to `g.mlist`. For example, you can generate several map copies and then delete them in one step:

```
g.copy vect=railroads@PERMANENT,myrailroads1
g.copy vect=railroads@PERMANENT,myrailroads2
g.copy vect=railroads@PERMANENT,myrailroads3
g.list vect
g.mremove vect="myrail*"
```

The module will collect the list of map names and ask for confirmation to delete. You want to double check, if any map is listed which you want to keep. You won't be able to undelete it.

Initially, you have access only to the MAPSET PERMANENT (read only) and your own MAPSET (read and write). If several MAPSETs exist for a given LOCATION, for example, when working within a team, you have to add these other MAPSETs to the MAPSET SEARCH PATH. Note that you have only read access to MAPSETs belonging to other users. We recommend using the module `g.mapsets` interactively, that is, starting it without parameters. For example, to add MAPSET `user2`, type + and a number listed along the name of the MAPSET that you want to add at the `new list>` prompt:

```
g.mapsets

Your mapset search list:
user1 <1>, PERMANENT <2>,

Available mapsets:
 1 user1    2 PERMANENT  3 user2
[...]
new list> + 3
```

You can restrict others' access to your own MAPSET through the use of the `g.access`. MAPSETs to which access is restricted can still be listed in another's MAPSET SEARCH PATH; however, access to these MAPSETs will remain restricted. To modify data from another user's MAPSET, copy them to your MAPSET using `g.copy`.

A useful command for getting information about the projection parameters and projection units for the LOCATION is `g.projinfo`. You may try it in the Spearfish LOCATION to get the coordinate system information.

LOCATION management. To copy a LOCATION or even a complete GRASS database directory, we recommend packaging the directories and extracing them in the destination directory. For example, to package the Spearfish LOCATION, enter:

```
cd /usr/local/share/grassdata
tar cvfz myspearfish_location.tar.gz spearfish
mv myspearfish_location.tar.gz  target_directory/
cd target_directory/
tar xvfz myspearfish_location.tar.gz
```

The target directory may be located on another machine, in this case you will transfer the file `myspearfish_location.tar.gz` on floppy/CD-ROM or through network to the destination machine and extract it there.

To remove a LOCATION from the GRASS database you have to change to the database directory:

```
cd /usr/local/share/grassdata
rm -r spearfish
```

This will remove the entire directory. If you want to avoid the delete confirmation prompts for every file/directory, add the flag `-f` to the `rm` command. Of course you can also use a file manager.

3.2. STARTING GRASS WITH A NEW PROJECT

When starting a new project, we need to define a new LOCATION and its projection and coordinate system.

```
------------------------------------------------------------
                          GRASS 5.3
LOCATION:
 This is the name of an available geographic location.
 is the sample data base for which all tutorials are written.
MAPSET:
 Every GRASS session runs under the name of a MAPSET.Associated
 with each MAPSET is a rectangular COORDINATE REGION and a list
 of any new maps created.
DATABASE:
 This is the unix directory containing the geographic databases

 The REGION defaults to the entire area of the chosen LOCATION.
 You may change it later with the command: g.region
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
LOCATION:   spearfishLL___  (enter list for a list of locations)
MAPSET:     user1_________  (or mapsets within a location)
DATABASE:   /usr/local/share/grassdata_____________

     AFTER COMPLETING ALL ANSWERS, HIT <ESC><ENTER> TO CONTINUE
                      (OR <Ctrl-C> TO CANCEL)
------------------------------------------------------------
```

Figure 3.5. GRASS text-based startup screen for selection of LOCATION, MAPSET and DATABASE

If we have data in different coordinate systems we have to import and store them in different LOCATIONS. However, the data can be re-projected between these LOCATIONS using the GRASS projection modules.

3.2.1 Latitude-Longitude

For illustration we create a new Spearfish LOCATION in latitude-longitude coordinate system. We assume that you have already created a directory for the GRASS DATABASE called `/usr/local/share/grassdata` (see Section 3.1.3). To create the new LOCATION, we first start GRASS:

```
grass53
```

GRASS starts with the TclTk interface that allows you to select your LOCATION and MAPSET. Because we want to define a new project we select "Create new" which brings us to the classic, non-graphical startup screen (see Figure 3.5).

For LOCATION enter the name for your new project (in our case `spearfishLL`), for MAPSET you can enter your name, and for DATABASE you should have `/usr/local/share/grassdata` (if it is not there, type it in). Note that this is an old fashioned interface, and when you want to change something, you need to type over it (BACKSPACE will not erase it). Once you have entered the new LOCATION, MAPSET, and DATABASE, you can continue with <ESC><ENTER>. Because your LOCATION does not exist yet, the following menu appears:

```
LOCATION <spearfishLL> - doesn't exist

Available locations:
----------------------
spearfish
----------------------
Would you like to create location <spearfishLL> ? (y/n)
```

Type `y` and you will get the following message:

```
To create a new LOCATION,you will need the follow. information:

1. The coordinate system for the database
        x,y (for imagery and other unreferenced data)
        Latitude-Longitude
        UTM
        Other Projection
2. The zone for the UTM database
   and all the necessary parameters for projections other than
   Latitude-Longitude, x,y, and UTM
```

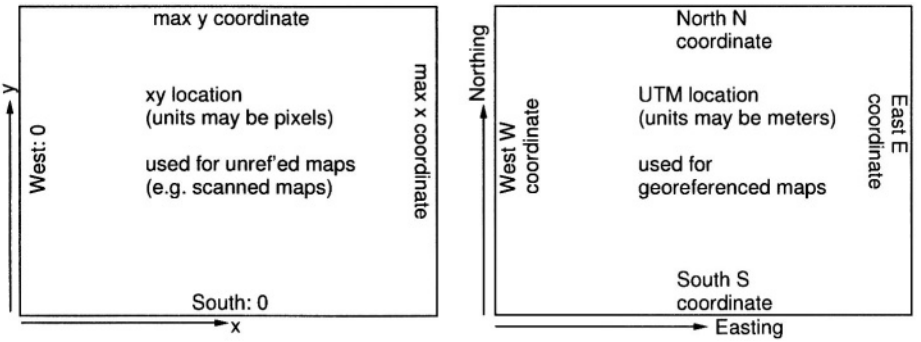


Figure 3.6. Definition of a xy LOCATION and of a projected LOCATION

```
3. The coordinates of the area to become the default region
   and the grid resolution of this region
4. A short, one-line description or title for the location

Do you have all this information for location <spearfishLL>? y
```

From the previous sections, you should understand what latitude-longitude or UTM means and you should know, based on the data that you want to work with (or from your supervisor, customer or instructor), what coordinate system you are going to use (see Figure 3.6 for a general idea). You can type again `y` and you will be asked to specify the new coordinate system:

```
A    x,y
B    Latitude-Longitude
C    UTM
D    Other Projection
```

Type the appropriate letter, in our example it will be `B` for Latitude-Longitude. We accept and continue with:

```
Please enter a one line description for location <spearfishLL>
> Spearfish Latitude-Longitude WGS84
ok? (y/n) [y] y

Do you wish to specify a geodetic datum for this location? y
Please specify datum name
Enter 'list' for the list of available datums
or 'custom' if you wish to enter custom parameters
Hit RETURN to cancel request
>list
Short Name      Long Name / Description
---
agd66   Australian_Geodetic_Datum_1966
                        (australian ellipsoid)
```

```
[...]
wgs84    World_Geodetic_System_1984
                          (wgs84 ellipsoid)
---
Please specify datum name
Enter 'list' for the list of available datums
or 'custom' if you wish to enter custom parameters
Hit RETURN to cancel request
>wgs84

Now select Datum Transformation Parameters
Enter 'list' to see the list of available Parameter sets
Enter the corresponding number, or <RETURN> to cancel request
>list
Number  Details
---
1       Used in Default wgs84 region
        (PROJ.4 Params towgs84=0.000,0.000,0.000)
        Default 3-Parameter Transformation
---
Now select Datum Transformation Parameters
Enter 'list' to see the list of available Parameter sets
Enter the corresponding number, or <RETURN> to cancel request
>1
```

These are all required parameters for Latitude-Longitude. Next you will be prompted to define your default region by defining the boundary coordinates of the project area and the default raster resolution (here we use sexagesimal degree notation):

```
              DEFINE THE DEFAULT REGION

            ====== DEFAULT REGION =======
            | NORTH EDGE: 44:30:06N_    |
            |                           |
WEST EDGE   |                           |EAST EDGE
103:52:14W_|                            |103:37:46W_
            | SOUTH EDGE: 44:22:23N_    |
            ============================

PROJECTION: 3 (Latitude-Longitude)         ZONE: 0

                GRID RESOLUTION
                   East-West:     0:00:01_____
                  North-South:    0:00:01_____

AFTER COMPLETING ALL ANSWERS, HIT <ESC><ENTER> TO CONTINUE
               (OR <Ctrl-C> TO CANCEL)
```

The default raster resolution (GRID RESOLUTION) is arbitrary, because you can change it later based on the needs of your application. For Latitude-Longitude LOCATIONs, you have to define the resolution in degree/minutes/seconds as well. You can leave this screen with <ESC><ENTER> and then check the list of parameters that appears:

```
  projection:    3 (Latitude-Longitude)
  zone:          0
  north:        44:30:06N
  south:        44:22:23N
  east:         103:37:46W
  west:         103:52:14W

  e-w res:      0:00:01
  n-s res:      0:00:01

  total rows:              463
  total cols:              868
  total cells:         401,884

Do you accept this region? (y/n) [y] >  y
LOCATION <spearfishLL> created!
Hit RETURN -->
```

If everything is correct, type y and RETURN and you will get back to the startup screen. Type <ESC><ENTER> again and you will get the message that your MAPSET does not exist yet (note that the MAPSET PERMANENT was created automatically):

```
Mapset <<user1>> is not available

Mapsets in location <>
----------------------
(+)PERMANENT

note: you only have access to mapsets marked with (+)
----------------------
Would you like to create < user1 > as a new mapset? (y/n) y
```

Type y and your new LOCATION with your MAPSET are created and GRASS prompt appears. You are now working in GRASS. You can check the definition of your LOCATION by running:

```
g.projinfo -p
g.region -p
g.region -m
```

The last command prints the geodesic resolution of the region in meters. Now the LOCATION is ready, and you can start importing data.

3.2.2 Universal Transverse Mercator

In this section we want to define a LOCATION with the UTM coordinate system, again for the Spearfish region, but now with NAD83 datum and the related GRS80 ellipsoid. To create the new LOCATION, we first start GRASS:

```
grass53
```

GRASS starts with the TclTk interface that allows you to select your LOCATION and MAPSET. Because we want to define a new project we select "Create new" which brings us to the classic, non-graphical startup screen (compare Figure 3.5, dialog shortened here):

```
                              GRASS 5.3
[...]
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
LOCATION:   spearfishNAD83  (enter list for a list of locations)
MAPSET:     user1_________  (or mapsets within a location)
DATABASE:   /usr/local/share/grassdata____________

     AFTER COMPLETING ALL ANSWERS, HIT <ESC><ENTER> TO CONTINUE
                      (OR <Ctrl-C> TO CANCEL)
```

For LOCATION enter the name for your new project (in our case `spearfishNAD83`), for MAPSET you can enter your name and for DATABASE you should have `/usr/local/share/grassdata`. Once you have entered the new LOCATION, MAPSET and DATABASE, you can continue with <ESC><ENTER>. Because this LOCATION does not exist yet, the following menu appears:

```
LOCATION <spearfishNAD83> - doesn't exist

Available locations:
----------------------
spearfish spearfishLL
----------------------
Would you like to create location <spearfishNAD83> ? (y/n)
```

Type `y` and you will get the following message (here slightly modified):

```
To create a new LOCATION,you will need the follow. information:

1. The coordinate system for the database
        x,y (for imagery and other unreferenced data)
        Latitude-Longitude
        UTM
        Other Projection
2. The zone for the UTM database and all the necessary
   parameters for project. other than Lat.-Long., x,y, and UTM
```

```
3. The coordinates of the area to become the default region
   and the grid resolution of this region
4. A short, one-line description or title for the location

Do you have all information for location <spearfishNAD83>? y
```

You can again type `y` and you will be asked to specify the new coordinate system:

```
A   x,y
B   Latitude-Longitude
C   UTM
D   Other Projection
```

Type the appropriate letter, in our example it will be `C` for UTM. Note that the following sequence of questions will vary for different coordinate systems that require different parameters, as we will show in some additional examples:

```
Please enter a one line descript. for location <spearfishNAD83>
> Spearfish UTM/NAD83
ok? (y/n) [y]

Do you wish to specify a geodetic datum for this location? y

Please specify datum name
Enter 'list' for the list of available datums
or 'custom' if you wish to enter custom parameters
Hit RETURN to cancel request
>list
Short Name      Long Name / Description
---
agd66   Australian_Geodetic_Datum_1966
                        (australian ellipsoid)
[...]
Hit RETURN to cancel request
>nad83

Now select Datum Transformation Parameters
Enter 'list' to see the list of available Parameter sets
Enter the corresponding number, or <RETURN> to cancel request
>list
Number  Details
---
1       Used in Florida
        (PROJ.4 Params nadgrids=FL)
        Transforms 'Old NAD83' to 'HPGN NAD83'
[...]
6       Used in Default nad83 region
        (PROJ.4 Params towgs84=0.000,0.000,0.000)
        Default 3-Parameter Transformation
---
```

```
Now select Datum Transformation Parameters
Enter 'list' to see the list of available Parameter sets
Enter the corresponding number, or <RETURN> to cancel request
>6

Enter Zone: 13
Is this South Hemisphere (y/n) [n] n
```

These are all the parameters needed for UTM (the system knows the other parameters like the valid ellipsoid GRS80). Next, you will be prompted to define your default region by defining the boundary coordinates of the project area and the default raster resolution:

```
              DEFINE THE DEFAULT REGION

            ====== DEFAULT REGION =======
            | NORTH EDGE: 4928040___     |
            |                            |
WEST EDGE   |                            |EAST EDGE
589980_____|                            |608940_____
            | SOUTH EDGE: 4914510___     |
            ============================

PROJECTION: 1 (UTM)                          ZONE: 13
                GRID RESOLUTION
                   East-West:     30________
                  North-South:    30________

AFTER COMPLETING ALL ANSWERS, HIT <ESC><ENTER> TO CONTINUE
               (OR <Ctrl-C> TO CANCEL)
```

The default raster resolution (GRID RESOLUTION) is arbitrary, because you can change it later based on the needs of your application. However, it is useful to choose a meaningful number, for example, based on the resolution of data that you want to import or the resolution that you plan to use in your work. In our example, we have chosen 30 meters. This resolution does not affect the vector and site data, which are stored with precise coordinates. Also, every raster map may have its own resolution (see more about raster data resolution in Chapter 5). You can leave this screen with <ESC><ENTER> and then check the list of parameters that appears:

```
projection:   1 (UTM)
zone:         13
north:        4928040
south:        4914510
east:         608940
west:         589980
```

```
  e-w res:      30
  n-s res:      30

  total rows:              451
  total cols:              632
  total cells:         285,032

Do you accept this region? (y/n) [y] >  y
LOCATION <spearfishNAD83> created!
Hit RETURN -->
```

If everything is correct, type y and RETURN and you will get back to the startup screen. Type <ESC><ENTER> again and you will get the message that your MAPSET does not exist yet (note that the MAPSET PERMANENT was created automatically):

```
Mapset <<user1>> is not available

Mapsets in location <>
----------------------
(+)PERMANENT

note: you only have access to mapsets marked with (+)
----------------------
Would you like to create < user1 > as a new mapset? (y/n) y
```

Type y and your new LOCATION with your MAPSET are created and GRASS prompt appears. You are now working in GRASS. You can check the definition of your LOCATION by running:

```
g.projinfo -p
g.region -p
g.region -l
```

which gives you the projection, coordinate system and units information that you have defined as well as the minimum and maximum coordinates and resolution in the LOCATION coordinate system with the flag -p or in geographic coordinates with the flag -l. Now you can start working on your project, by importing some data as explained in the chapters about raster, vector and site data processing.

3.2.3 State Plane

As we have mentioned, the dialog used for the LOCATION definitions will vary depending on the coordinate system. We will illustrate it by the following example, where we use the State Plane Coordinate System. First we need to find out the coordinates of our study region as well as the ellipsoid and datum information (e.g., from the metadata of the file that we plan to import). Then

we can start GRASS, provide the name of the new LOCATION, in our example, `wake-spm` and MAPSET, e.g., `user1`. Then type <ESC><ENTER> and after going through the previously described steps, you will be prompted for coordinate system. Select D for other and proceed as follows:

```
Please specify the coordinate system for location <spearfishSPF>
A   x,y
B   Latitude-Longitude
C   UTM
D   Other Projection
RETURN to cancel
> D
[...]

Please enter a one line description for location <spearfishSPF>
> Spearfish State Plane Feet NAD27
[...]

Please specify projection name
Enter 'list' for the list of available projections
Hit RETURN to cancel request
> list
ll -- Lat/Lon
utm -- Universe Transverse Mercator
stp -- State Plane
aea -- Albers Equal Area
[...]
```

Based on a list of supported projections, select `stp` for State Plane and then we continue by providing the rest of the parameters similarly, using `list` to get the supported options and then selecting the one that applies to our case (we show only the question and answer here, State FIPS code for South Dakota is 46, County FIPS code for Lawrence county is 81):

```
>stp

Do you wish to specify a geodetic datum for this location? [y]
Please specify datum name
>nad27

Now select Datum Transformation Parameters
>10

Specify State FIPS (numeric) code
>46
You have chosen state SD, Correct(y/n) [y] y

Specify County FIPS (numeric) code for state SD
>81
You have chosen LAWRENCE county, correct(y/n) [y] y
```

```
Specify State Plane 1927 or 1983
Enter '27' or '83'
Hit RETURN to cancel request
>27

Specify the correct units to use:
Enter the corresponding number
1.      US Survey Foot (Default for State Plane 1927)
2.      International Foot
3.      Meter
>1
```

After providing the units you will get to the DEFINE THE DEFAULT REGION screen (see our previous example). Now provide the coordinates of the north, south, west and east edge of your project area (in our case N: 271680, S: 195240, W: 977640, E: 1068000) and a suitable resolution (for example 30 m, remember, that you may change it for your work any time using `g.region`):

```
projection: 2 (State Plane)
zone:       0
datum:      nad27
ellipsoid:  a=6378206.4 es=0.006768657997291094
north:      271680
south:      195240
west:       977640
east:       1068000

e-w res:      30
n-s res:      30

total rows:       2548
total cols:       3012
total cells: 7,674,576

Do you accept this region? (y/n) [y] > y
```

After approving your region parameters and new MAPSET creations (see the previous example) you will get the GRASS prompt and you can start working with your Spearfish data in State Plane coordinate system.

3.2.4 Non-georeferenced xy coordinate system

If you need to work with non-georeferenced data or you do not know the parameters of your coordinate system, or your coordinate system is not supported by GRASS, you can define a LOCATION in a general, non-georeferenced coordinate system `xy`.

To define a new xy LOCATION, start `grass53` and enter new names for LOCATION and MAPSET; for example, `area-xy` and `user1`. Similarly to the procedure described in Section 3.2 proceed to the question "Please specify the coordinate system for location area-xy". The coordinate system we need here is `A` "x,y". After entering a one line description, you reach the LOCATION region definition screen. Now define the region size in x and y direction (rows and columns). It should cover at least the size of the image or map that you want to import. The xy LOCATION can be defined larger than needed because the actual memory used depends only on the size of your imported file. When working with imagery data, set the west and south values to 0 (zero) and the north and east values to the number of rows and columns of the image (or more, compare Figure 3.7). The GRID RESOLUTION can be set to 1, because the units are pixels. After leaving this menu and accepting the definition, the new LOCATION is created. You can return to the GRASS startup screen and leave it again to create the MAPSET and to enter GRASS.

Later on, in Section 9.2.1, we describe a method for automatic creation of a LOCATION from a raster data set.

3.3. COORDINATE SYSTEM TRANSFORMATIONS

Geospatial data for a given study area are often provided in different coordinate systems (for example, combination of the UTM, State Plane and geographic coordinates is quite common in USA). It is therefore important to have the capability to transform data between different projections and coordinate systems.

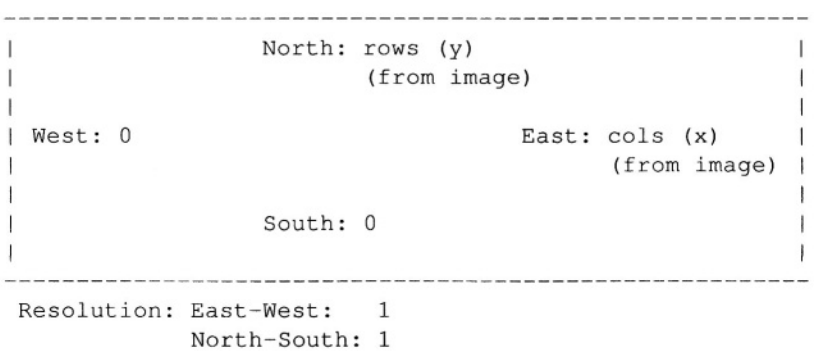


Figure 3.7. Definition of a region for xy LOCATION suitable for importing an image or scanned map. Units are pixels

GRASS and its projection support through PROJ4. The projection library in GRASS 5.3 is either the PROJ 4.3.3 developed by USGS (Evenden, 1995) or any later PROJ4 version used as an external library. The PROJ4 library is now maintained by volunteers[6] and contains a stand-alone program `cs2cs` for reprojection of coordinate lists. Use GRASS with a recent version of PROJ4 recommended, as it also supports datum transformations from version 4.4.5 onwards. The general procedure for transforming between two projections is (internally) always performed through geographical coordinates:

Projection 1 $\rightsquigarrow$ latitude-longitude $\rightsquigarrow$ Projection 2

The projection information in a GRASS LOCATION is stored in the PERMANENT MAPSET in files `PROJ_INFO` and `PROJ_UNITS`. The following parameters may appear depending on the actual projection: `proj` (projection type), `name` (projection name), `ellps` (ellipsoid), `a` (ellipsoid: equatorial radius), `es` (ellipsoid: eccentricity squared), `zone` (zone for the area), `unfact` (conversion factor from meters to other units, e.g. feet), `lat_0` (standard parallel), `lon_0` (central meridian), `k` (scale factor), `x_0` (false easting) and `y_0` (false northing).

To simplify the definition of a projection, PROJ4 provides support for EPSG (European Petroleum Survey Group) codes, aimed at standardization of common projection definitions. Projections and coordinate systems including geodetic datum can either be constructed or selected via EPSG code from predefined list entries. The list of EPSG codes is usually installed at `/usr/local/share/proj/epsg`.

Depending on the type of data that need to be transformed, transformations can be done in two ways:

- ASCII file coordinate lists can be transformed between any of the more than 120 supported projections by running the external command `cs2cs` provided by PROJ4;
- raster, vector and site map layers are transformed between two existing LOCATIONS with given coordinate systems using the commands `r.proj`, `v.proj` and `s.proj`.

3.3.1 Coordinates lists

PROJ4 provides the command `cs2cs` for reprojection of point lists given by coordinate pairs; for example, resulting from GPS or for map corners. On command line, the source and the target projections have to be defined. Subsequently, the coordinate pairs are queried or read from an ASCII file, transformed, and written either to the screen or redirected to an output file. The program also reports the built-in projections (p, extended list with P), ellipsoids (e), prime meridians (m), datums (d) and units (u):

```
cs2cs -lp
cs2cs -lP
cs2cs -le
cs2cs -lm
cs2cs -ld
cs2cs -lu
```

For example, to transform the corner points of the Spearfish LOCATION from UTM/Clark66/NAD27 to latitude-longitude/WGS84, we run (enter in one line):

```
cs2cs -v +proj=utm +zone=13 +ellps=clrk66 +datum=NAD27 \
      +units=m +to +proj=latlong
```

This command prints out the projection parameters (due to the `-v` flag) and then waits for input. Now you can type in UTM coordinate pairs, optionally with the related elevation. Entering for example the north/west map corner of the Spearfish LOCATION delivers:

```
589980 4928010
103d52'7.399"W  44d30'6.293"N 0.000
```

which represents the corresponding latitude-longitude/WGS84 coordinates. The command `cs2cs` performs the required datum transformation.

As other example, we transform coordinates from UTM/Clark66/NAD27 (Spearfish LOCATION) to LAEA/Sphere/no datum. We will store the values in a file. To get the LOCATION corner coordinates, start GRASS with the Spearfish LOCATION and run:

```
g.region  -pd
```

From the reported values we create a file `spearfishUTM_NAD27.txt` containing the following four corners. The input file must be written in plain ASCII format containing row-wise easting and northing:

```
589980 4928010
609000 4928010
609000 4913700
589980 4913700
```

Now we convert the coordinates in the file to the standard raster map projection of the National Atlas of the U.S.[7], which is Lambert Azimuthal Equal Area (LAEA) on a Sphere:

```
cat spearfishUTM_NAD27.txt | cs2cs -v +proj=utm +zone=13 \
  +ellps=clrk66 +datum=NAD27 +units=m  +to \
  +proj=laea +lat_0=45 +lon_0=-100 +x_0=0 +y_0=0 +ellps=sphere\
  +units=m > spearfishLAEA.txt
```

The command line will reproject the coordinate pairs stored in file `spearfishUTM_NAD27.txt` to coordinates in Lambert Azimuthal Equal Area on a Sphere without geodetic datum and write the result to file `spearfishLAEA.txt`:

```
-306676.27      -48094.86 0.00
-287744.36      -49261.79 0.00
-288620.30      -63563.34 0.00
-307552.56      -62397.59 0.00
```

3.3.2 Map layers

The projection of raster, vector and site map layers between two different coordinate systems requires two LOCATIONs: one LOCATION holding the source map layer and input coordinate system information, and another LOCATION for reading the target coordinate system information and storing the projected map layer.

We will illustrate the procedure using the following example. All maps such as the elevation raster map of the Spearfish region is available in UTM/NAD27 coordinate system (in the sample `spearfish` LOCATION, PERMANENT MAPSET). We want to transform this map into the `spearfishNAD83` LOCATION we have defined in Section 3.2.2. We now start GRASS with the `spearfishNAD83` LOCATION and "pull" the elevation map layer from the source LOCATION `spearfish` into our current LOCATION as follows:

```
r.proj in=elevation.dem location=spearfish \
       mapset=PERMANENT method=nearest
d.mon x0
d.rast elevation.dem
```

After some computation time, the source map is available projected in the current LOCATION and MAPSET. Per default, the resulting map is saved with same name. Note that the resolution and region (map extent) of the projected map layer depends on the current region settings in the target LOCATION `spearfishNAD83`, you can verify the settings with `g.region -p`. Like that you can also limit transformations to subregions with desired resolution. Please refer to the manual page of `r.proj` for the interpolation methods used during the transformation.

Similarly, you can project the vector and site map layers:

```
v.proj -s in=roads location=spearfish mapset=PERMANENT
d.vect roads
s.proj in=archsites location=spearfish mapset=PERMANENT
d.sites archsites
```

For the vector data, the entire map is always projected, because creation of subregions is not supported. Again, keep in mind that the included datum transfor-

mations (here from NAD27 to NAD83) are only supported from GRASS 5.3 onwards.

3.3.3 Reprojecting with GDAL/OGR tools

Map reprojection and spatial subsetting of GIS data sets is also possible outside of GRASS. Sometimes it is more convenient to reproject maps to a desired projection (or coordinate system, ellipsoid, geodetic datum) before importing it into a GRASS LOCATION. The free GDAL/OGR libraries[8] provide a set of tools to perform such map preprocessing. GDAL is the raster data engine, while OGR supports vector data.

GDAL and raster data. GDAL is a translator library for raster geospatial data formats. Several tools are provided, we introduce some of them here. The general command structure is:

```
gdalinfo --formats

gdalinfo [flags] rastmap
gdal_translate [flags] [parameters] inrastmap outrastmap
gdalwarp [flags] [parameters] inrastmap outrastmap
```

The following example shows how to convert a free LANDSAT-TM7 scene for the Spearfish region, as available from GLCF Maryland[9], from WGS84 to NAD27 to match the Spearfish LOCATION projection:

```
gdalinfo p033r029_7t20000712_z13_nn10.tif

#EPSG codes available in /usr/local/share/proj/epsg
#
#Reprojection from UTM/WGS84 to UTM/NAD27/Clarke66
#(use -tr xres yres to maintain original resolution):
gdalwarp -t_srs '+init=epsg:26713' -tr 28.5 28.5 \
         p033r029_7t20000712_z13_nn10.tif \
         p033r029_7t20000712_z13_nn10_NAD27.tif


#Cut out region of interest:
#  boundary coordinates    W      N       E      S
gdal_translate -projwin 589980 4928010 609000 4913700 \
               p033r029_7t20000712_z13_nn10_NAD27.tif \
               p033r029_7t20000712_z13_nn10_NAD27_small.tif
#Verification:
gdalinfo p033r029_7t20000712_z13_nn10_NAD27_small.tif
```

The preprocessed LANDSAT-7 band is ready now for import into the Spearfish LOCATION with `r.in.gdal`. You can also download the prepared data set from the book related Web page.[10]

OGR and vector data. Vector maps can be reprojected and preprocessed using the OGR library, with the SHAPE format as default output. We use the program `ogrinfo` to report vector map metadata and `ogr2ogr` to perform map reprojections. The general command structure is:

```
ogrinfo --formats

ogrinfo [flags] [parameters] vectmap [layer [layer ...]]
ogr2ogr [flags] [parameters] outvectmap invectmap [parameters]
```

In our OGR example we use the geological map from the National Atlas of the U.S. The map is provided in SHAPE format. We prepare it here for import into the Spearfish LOCATION which requires reprojection and spatial subsetting:

```
ogrinfo -summary geolgyp.shp geolgyp

#Assign missing projection information (create SHAPE .prj file):
# Vector maps of National Atlas of the U.S. are in
# latitude-longitude on a Sphere
ogr2ogr -a_srs '+proj=latlong +ellps=sphere' \
        geolgyp_LL_sphere.shp geolgyp.shp
#Verification:
ogrinfo -summary geolgyp_LL_sphere.shp geolgyp_LL_sphere
```

Now the SHAPE file contains the projection information. Next we reproject to UTM/NAD27/Clarke66 as required for the Spearfish LOCATION and cut out the region of interest. For convenience we use the EPSG code to select the output projection parameters. Note, that it is necessary to have the NAD datum shift files installed (see the PROJ4 web page, section "Frequently Asked Questions" for details):

```
#EPSG codes available in /usr/local/share/proj/epsg
#
#Reprojection from LatLong/Sphere to UTM/NAD27/Clarke66
#
#Cut out region of interest (boundary coords. W S E N):
ogr2ogr -t_srs '+init=epsg:26713  +nadgrids=ntv1_can.dat' \
        -spat  -103.87111 44.37293 -103.62943 44.50174 \
        geolgyp_UTM_NAD27_small.shp \
        geolgyp_LL_sphere.shp

#Verification:
ogrinfo -summary geolgyp_UTM_NAD27_small.shp \
        geolgyp_UTM_NAD27_small
```

The resulting SHAPE map can be imported into the Spearfish LOCATION.

NOTES

1 GRASS 5.3 users manual, `http://grass.itc.it/gdp/online.html`
2 FreeGIS Web portal, `http://www.freegis.org`
3 OpenOSX site, `http://openosx.com/products.html`
4 Source for Spearfish data set: `http://grass.itc.it` (section "sample data")
5 Description of Spearfish data set, `http://grass.itc.it/gdp/tutorial/spearDB.ps.gz`
6 PROJ4 Web site, `http://www.remotesensing.org/proj/`
7 National Atlas of the U.S. download area, `http://nationalatlas.gov/atlasftp.html`
8 GDAL library, `http://www.remotesensing.org/gdal/` OGR library, `http://www.remotesensing.org/gdal/ogr/`
9 GLCF Maryland LANDSAT data for Spearfish (SD) region, `ftp://ftp.glcf.umiacs.umd.edu/glcf/Landsat/WRS2/p033/r029/`
10 Spearfish LANDSAT-TM7 scene, spatial subset, reprojected to UTM/NAD27: `http://mpa.itc.it/grasstutor/`, data sets, file `p033r029_20000712_NAD27_small.tar.gz`

Chapter 4

GRASS DATA MODELS AND DATA EXCHANGE

GRASS stores the georeferenced data as raster, vector and site map layers. In this chapter, we explain the basic properties of GRASS data models and their management. The import and export for the various raster, vector and point data formats is presented, including a number of examples.

4.1. RASTER DATA

Raster data, stored in GRASS as a matrix of values, represent either a continuous field (surface), an image, or geometric objects (points, lines, areas) corresponding to discrete fields (Figure 4.1). For surfaces, the values in the matrix are assigned to the center points of grid cells. They represent the actual measured or computed values (e.g. elevation, slope, temperature). For non-continuous fields (images, geometric objects), the values are assigned to the entire cell area and represent the category number.

4.1.1 GRASS raster data model

A raster map layer is stored in GRASS as a set of files, which include the raster data and their description, organized as follows:

- generic matrix of values in a compressed format which depends on the raster data type (integer, floating point or 3D grid);
- map header which contains georeferencing data, resolution, number of rows and columns, range of values and histogram;
- optional category file which contains text or numeric labels;
- optional color table;

- optional timestamp;
- history file which contains metadata such as the data source or other information provided by the user;
- reclass table for a map that is a reclassification of another map.

All this information is stored in the relevant subdirectories in the LOCATION/MAPSET directory. In the following sections, we describe how these components are managed and queried.

In GRASS, raster data can be stored as a 2D integer grid, 2D floating point grid (single or double precision), or a 3D floating point grid (single or double precision). The internal GRASS raster data storage format is architecture independent and portable between 32 bit and 64 bit machines. As a result of that a GRASS DATABASE may be accessed in a heterogeneous network file sys-

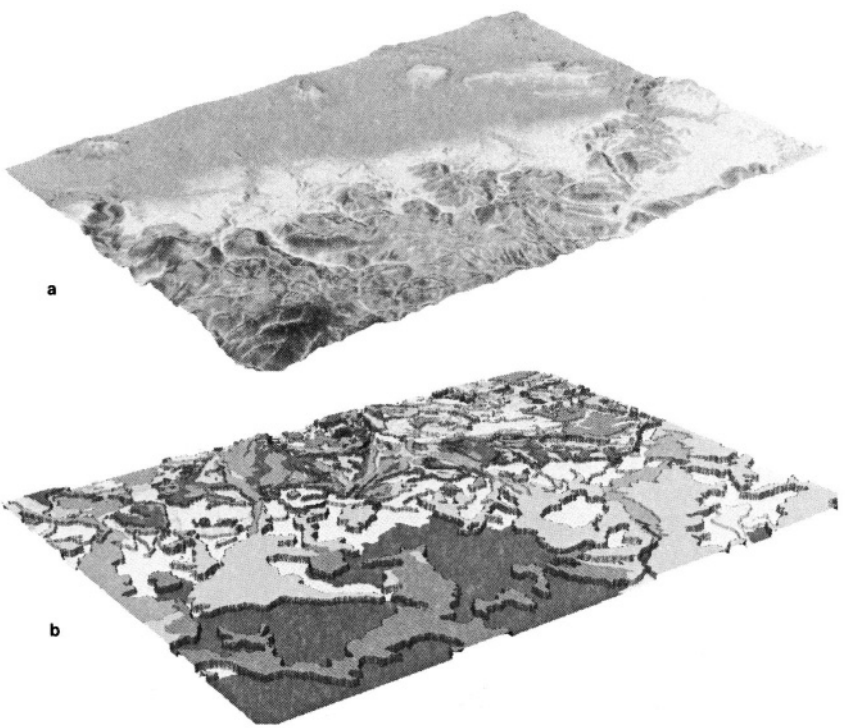


Figure 4.1. Types of raster data: a) continuous field, b) discrete areas

tem (NFS) without compatibility problems for raster data (from GRASS 5.7 onwards vector data is portable across different architectures). Internally, the integer format is called CELL, single precision floating point is called FCELL, and double precision floating point is DCELL.

The choice of the integer or floating point data depends on the user's application. Their use can be described in general as follows:

Integer raster map layers: Rasterized geometric objects (points, lines, areas) are represented by non-continuous (discrete) fields. Each raster cell is assigned an integer value called *category number*. Each of the categories may have a label (usually a character string but a number can be used as well) describing the meaning or properties of these categories. Such category data as well as reclassified data and image data are stored in integer format (GRASS CELL type).

Floating point raster map layers: Continuous fields such as elevation surfaces are often stored as floating point data (GRASS FCELL and DCELL types). It is possible to label these data by defining ranges of values (which can be interpreted as classes) and assigning each range a label (text or number).

3D floating point raster map layers: Raster volumes are stored as a voxel (volume pixel) data model (GRASS GRID3D type) designed to support representation of trivariate continuous fields.

Note that continuous field data can be represented in integer format (for example, some digital elevation models). This is a limitation of the data quality, and such data should be treated as continuous field representations! We will point out the related specific issues depending on the application later in this chapter and in Chapter 12.

GRASS also allows the user to create raster map layers by re-defining the classes as described in Section 5.1.5. In such a case, the reclassified map layer does not contain any data, but serves as a reference to another map layer along with a reclass table that is used to reclassify the values of the referenced raster map. From the user's point of view such a map behaves as a regular raster map. Few GRASS modules do not work with reclassed maps; in such a case the module will report an error and suggest that the user generates a true copy of such a map (see Section 5.1.3).

4.1.2 Managing raster map resolution and boundaries

GRASS differs from other systems in the way it handles region (map extent) and resolution. While each raster map layer has its own resolution defined in its header, the operations with raster data are performed using the "working"

(or current) region and a resolution set by `g.region`. If the current region is smaller than the map extent of the raster that is being processed, the operation is applied only to the subset of the raster file defined by the current region. If the resolution is different, the raster is automatically resampled (see Section 5.3.4). This approach is also used when exporting raster data. It makes exporting subsets of raster maps very convenient, including export at a lower resolution. Note that the GRID RESOLUTION defined when setting up a LOCATION is the default region resolution and will be used only if the current region is set to the default region.

To adjust the current region to different values, you can use `g.region`. After starting the module, you get to the menu which allows you to modify the current region boundaries and resolution. If necessary, you can save the current region settings as a region file. This is sometimes useful when working on different subregions within the given LOCATION. This module can be efficiently used in the command line mode, for example:

```
g.region res=12.5
```

will set the resolution to 12.5 map units (e.g. meters). The region can be also defined from existing maps:

```
g.region rast=elevation.dem -p
```

which will adjust the current region according to `elevation.dem`. The flag `-p` additionally prints the current settings to the screen as follows (UTM projection):

```
projection: 1 (UTM)
zone:       13
datum:      nad27
ellipsoid:  clark66
north:      4928000
south:      4914020
west:       590010
east:       609000
nsres:      30
ewres:      30
rows:       466
cols:       633
```

If you want to reset to the default region (boundary coordinates and raster resolution) of the LOCATION use the `-d` flag:

```
g.region -dp
```

While the 3D capabilities are still in the development stage, we should mention at least briefly management of boundaries and resolution for volume data.

The 3D region is managed with `g3.region` or, with its command line version, `g3.setregion`, similar to 2D GRASS regions. If no 3D region exists yet, it must be created with `g3.createwind`. This module extends the 2D region definition by the third spatial dimension along with a user defined voxel resolution.

4.1.3 Import of georeferenced raster data

When importing raster data, we need to distinguish three general raster format types:

- Binary image formats, which include only positive integer values (e.g., JPG, PPM, PNG etc.);
- Binary raster formats: integer and floating point supported, both negative and positive values, single and multiple bands, single and multiple resolutions (such as ERDAS IMG, HDF, GeoTIFF etc.);
- ASCII raster formats, which can have integer and floating point values, both negative and positive (e.g., ARC-ASCII, ASCII-GRID, GRASS-ASCII etc.).

All common GIS raster formats are supported. Note that only a few of them (e.g., ASCII) handle negative and floating point values. When obtaining data, make sure to get information about the coordinate system (projection, datum, etc.). For some formats, the metadata are directly stored in support files which are read when importing the data.

Most raster maps can be imported with `r.in.gdal`. It requires the GDAL library[1] which is included in the GRASS binary releases. It supports a wide (and growing) range of formats and is able to auto-detect them. When importing with this command, you can automatically extend the LOCATION definition by using the flag `-e` in case that the imported map is larger than the default region:

```
r.in.gdal -e in=d44103d7.tif out=d44103d7
```

If your data set does not contain projection information and you are sure than the data projection matches the projection of the LOCATION, the `-o` flag allows you to use the LOCATION projection information for the imported map.

When using the `tcltkgrass` user interface select: "IMPORT" ⇝ "RASTER MAP" ⇝ "Various formats". The opened window contains a button "file" which provides a small file manager for selecting the source file. A new name which represents the name in GRASS DATABASE has to be typed into the second line.

After a successful import, it is useful to run the module `r.support` with `-r` flag on the recently imported data set. It will calculate statistical data, such as the range, which is required by other modules:

```
r.support -r d44103d7
```

The import of multispectral satellite data is explained in Section 9.2.1.

Generating a new LOCATION from an external raster map. The module `r.in.gdal` provides an additional, very useful functionality by automatically generating a LOCATION from an external raster data set. For this purpose, it has to be run within another LOCATION (this LOCATION can be completely unrelated to the imported data and its setting won't be affected by `r.in.gdal` execution). For example, you can import a new 10 m DEM for Spearfish area, provided as ArcGRID coverage `09233536`[2] in geographic coordinate system (NAD83 datum), and at the same time create a new LOCATION `spearfishllnad83` that is defined with the parameter `location`. The projection information is taken from the input data set, in our case stored in the file `09233536/PRJ.ADF`. To import the DEM run the `r.in.gdal` command from the `spearfish` LOCATION that we have created in Section 3.2.1:

```
unzip 09233536.zip
cd 09233536/
r.in.gdal 09233536 location=spearfishllnad83 output=ned10m.ll
```

To display the imported DEM, exit GRASS and start it with new `spearfishllnad83` LOCATION. You should see your imported file `ned10m.ll` when you run `g.list rast` and check the coordinate system information using `g.projinfo`. Note that the datum is NAD83, therefore we could not directly import this map into the `spearfishLL` LOCATION used in Section 3.1.3 that uses datum NAD27.

Generally, if no projection information is present, the new LOCATION will not have the coordinate system definitions. The module `g.setproj` can then be used within the new LOCATION to generate the projection information. Be careful to use `g.setproj` only in a new LOCATION! The module does not perform any coordinate transformation of data (see Section 3.3 to learn how to do that).

Import of TIFF raster files. Data in the TIFF/TFW format usually consist of two files: `your_maptif` and `your_map.tfw`. Make sure to get both files when obtaining data. For GeoTIFF format it will be a single file which may also contain projection information and one or several maps. Using the module `r.in.gdal` it is quite easy to import such a data set:

```
r.in.gdal -e in=your_map.tif out=your_map
```

The flag `-e` allows you to automatically extend the LOCATION DEFAULT_WIND based on new data set, if the imported map is larger. If the imported TIFF image consists of several bands, they are extracted respectively into the current MAPSET. This will usually occur when importing aerial color images which are delivered in RGB (red, green, blue) channels. It may happen that the data set does not contain the projection information. The module will not import the file unless you use the `-o` flag (override). In this case you are using the projection information of the current LOCATION which makes sense when the map belongs to this LOCATION and it is in the identical coordinate system.

Import of ASCII raster files. Raster data in ASCII format can be in different GIS formats. Besides the GRASS ASCII raster format (supported by `r.in.ascii` and `r.out.ascii`), the ARC/INFO ASCII GRID format is commonly used. It can be imported by `r.in.gdal`. Data in ARC/INFO ASCII GRID sometimes have an associated `map.prj` file which contains projection information. If not available, the `-o` flag in `r.in.gdal` allows us to use the current projection information from the LOCATION.

Another method to generate raster area or lines from given coordinates is supported by `r.in.poly`. The module accepts text files containing coordinate pairs with labels. Either raster area ("A") or raster line ("L") type can be specified. An example for a single area (store next code as text file `rasterarea.txt`, we use UTM coordinates for Spearfish region):

```
A
591316.80   4926455.50
591410.25   4926482.40
591434.60   4926393.60
591341.20   4926368.70
= 42 stadium
```

It is important to define the raster resolution before importing this "vector" file (e.g. 1 meter raster resolution). The import of this file with `r.in.poly` will generate a raster area with given corner points and labeled as "42 stadium":

```
g.region res=1
r.in.poly in=rasterarea.txt out=stadium
r.info stadium
```

The resulting map contains the desired stadium area.

Import of ARC/INFO Binary GRID files. The ARC/INFO Binary GRID coverages can be imported directly using `r.in.gdal`, as shown above (see paragraph about generating a new LOCATION). As an input, you can either specify the grid coverage directory or the `w001001.adf` grid file inside the grid directory.

Import of USGS DOQ flies. It is also possible to import DOQ (Digital Orthophoto (Quarter) Quadrangles) data from USGS using `r.in.gdal`. The included projection information is respected as well as other metadata. Be sure to avoid the DOQ data in MrSID format because it is proprietary and therefore not supported. Conversion with the proprietary converter is done as follows (but it will not transfer projection information):

```
%#but it will not transfer projection information:
mrsiddecode -tif -input i44103d7_a.sid -output i44103d7_a.tif

#verification:
gdalinfo i44103d7_a.tif
```

As explained in Section 3.3.3, the missing projection information can be reassigned with `gdal_translate`.

Import of binary arrays: GTOPO30 DEM, Etopo-5 DEM, Globe DEM, BIL, AVHRR and GMT flies. The module `r.in.bin` reads numerous binary array grids such as GTOPO30 DEM (worldwide elevation data in 30 arc-seconds resolution, USGS), Etopo-5 DEM (worldwide elevation data in 5 minutes resolution), Globe DEM (worldwide elevation data in 30 arc-seconds resolution, NOAA), BIL, AVHRR (Advanced Very High Resolution Radiometer) and GMT (Generic Mapping Tool). Please refer to the related manual page (`g.manual r.in.bin`) for encoding details. Examples are:

- import of GTOPO30 DEM data (you can add `anull=-9999` if you want the sea level be set to NULL):

```
r.in.bin  -s input=E020N90.DEM out=gtopo30 bytes=2\
    north=90 south=40 east=60 west=20 r=6000 c=4800
```

- import of a GMT type 1 (float) binary array (-b may be used to swap bytes if required):

```
r.in.bin -hf input=your_map.grd out=gmtmap
```

- import of a AVHRR image (here the raster map will be assigned a north=128, south=0, east=128, west=0 as rows and cols are defined):

```
r.in.bin in=p07_b6.dat out=avhrr c=128 r=128
```

Import of DTED files. The module `r.in.gdal` supports import of DTED (Military Terrain Elevation Data) at Levels 1 and 2. The DTED data include projection information which is respected.

Import of USGS SDTS DEM flies. The USGS SDTS DEM data sets consist of a number of files. Again, use `r.in.gdal` to import such a data set. Each DEM should have one file with a name like `mapcatd.ddf` which has to be specified as the import file. Projection and georeferencing information is respected.

Import of raster files in common image formats. You can directly import raster maps and images in the following formats: PNG (Portable Network Graphics), PNM (netpbm), uncompressed GIF, TIFF and JPEG. The module `r.in.gdal` will read these formats. As most of them do not provide projection information, you will have to apply the georeferencing manually. See the next paragraph how to do that.

Import of raster data without ancillary georeferencing flies. If you obtain a raster map in a common format such as TIFF, but without the related TFW file, you can update the geocoding manually. Of course you have to get the related georeference information from the data provider.

First, import the map with `r.in.gdal`. The lower left corner coordinates of the imported map will be at the origin of the LOCATION coordinate system, which is usually outside the study area. To georeference the map, the information in its header needs to be modified using the module `r.support`. After starting it, specify the map name. Then go through following dialog:

1 "Edit the header?" `y`. Rows and columns can be checked now. The values should be correct.

2 Pressing <ESC><ENTER> changes into the coordinates menu which looks similar to the LOCATION definition screen.

3 Now you have to update the boundary coordinates. Enter the correct coordinates and GRID RESOLUTION for this map by moving around with the cursor keys. Afterward hit <ESC><ENTER> to proceed.

4 The additional questions can be skipped with <ENTER>.

Further information on capabilities of the module `r.support` (e.g. changes of the color map) can be found in the GRASS manual of `r.support`.

4.1.4 Import and geocoding of scanned maps

In this section we explain rectification and georeferencing of a scanned map. For this procedure, it is important to be aware of the relation between geometrical length, scale and spatial extension. This general cartographical relation is also valid when transforming an analog map into a digital map. Because you will most likely be using a scanner (at least to create a backdrop map for the

vectorization within the GIS when you don't have access to a digitizer board), these terms are of great importance. Also, keep in mind the proper handling of copyright restrictions when scanning maps.

Determining scanning parameters. The relationship between distance in "nature" (ground truth) and corresponding length of a raster cell is determined by the scanning resolution. When working with toposheets, scanning resolution somewhere between 150 and 300 dpi is recommended. Of course the text labels on the map should stay readable. Depending on the number of colors in the map, the image can be scanned as color image with 256 colors. As an example let us assume a scanning resolution of 300 dpi. First we calculate the equivalent in centimeters:

$$300dpi = 300\frac{lines}{2.54cm} = 118.11\frac{lines}{cm} \tag{4.1}$$

Suppose that the scale of the scanned map is 1:25,000. Thus, one centimeter on the map is equivalent to 25,000 cm in nature. Now we can calculate the distance in nature corresponding to the length of a raster cell:

$$\frac{distance\ in\ nature}{scanned\ lines\ per\ cm} = \frac{25,000cm}{118.11lines} = 211.6\frac{cm}{line} = 2.12\frac{m}{line} \tag{4.2}$$

The resulting value of 2.12 m is the spatial resolution of the map at the 300 dpi scan resolution. If you want the spatial resolution to be an integer, do the inverse calculation and adjust the scanning resolution accordingly.

Geocoding of scanned maps. After scanning the map, we store it in an external file. If needed, we can convert it to a GRASS supported format using `gimp`, `display` or `xv` which are available for many operating systems, as well as the netpbm tools[3] which can be run on command line.

To geocode a scanned map, we first import it into a temporal xy LOCATION, assign related coordinates to a few (usually 4) known points, and rectify it into a target LOCATION using a specific GRASS module. Note that the projection of the target LOCATION must be identical to that of the scanned map. We recommend getting the four points from the paper map. The general idea is shown in Figure 4.2. We will be using image processing tools for geocoding which are also explained in Section 9.4.1.

First, we have to create a xy LOCATION with a region large enough for the imported map. If you do not know the number of rows and columns of the scanned map, you can find it using one of the above mentioned image viewers. Start GRASS and define a new xy LOCATION (rows and columns according to the size of the scanned map, GRID RESOLUTION is 1 pixel). It does not matter if you define the xy LOCATION larger; unused cells will not affect the

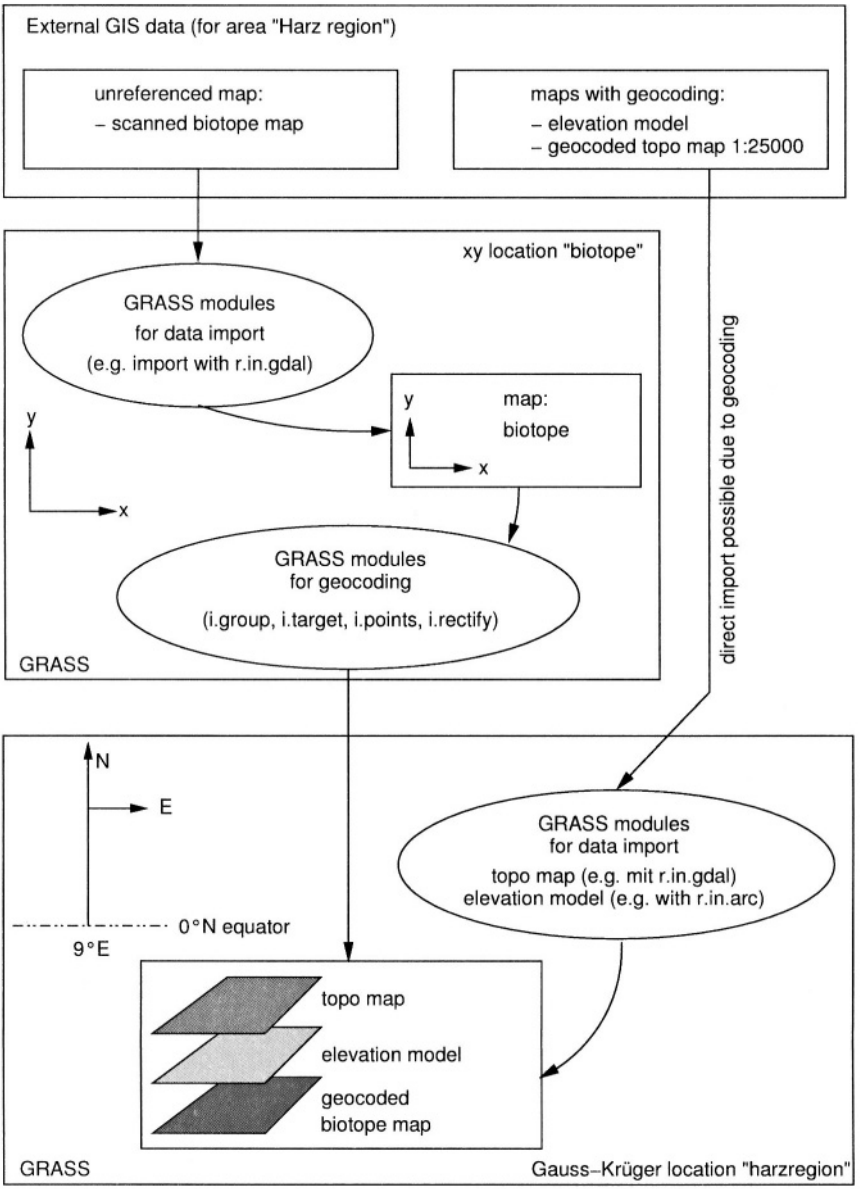


Figure 4.2. Sample workflow to import GIS data and to geocode scanned maps

allocated space on the hard drive. Now you can import the map using, for example, `r.in.gdal`.

After importing the map, the process is as follows (we need to use some commands here that will be the main topic in Chapter 9). The scanned map has

to be added to a so-called "image group". This is simply a list of raster maps to be processed which is required to work with the `i.*` (image) modules. In our case, we only add this single map to the group list. To set up such an image group, run `i.group` either on the command line or interactively:

- enter a name for this group, for example: `mapscan`. Hit <ESC><ENTER> to reach the main menu and confirm the new group name;
- mark the scanned map with a `x`; hit <ESC><ENTER> to exit;
- leave the module with <ENTER>.

The next step is to define a target LOCATION (for example in UTM). For this purpose run `i.target`: select the group and enter the name of LOCATION and MAPSET (use `list`, <ESC><ENTER> to get a list of available LOCATIONS and MAPSETS).

After having successfully set the image group and the target LOCATION, we now define the geographic reference points. They have to be set to "tell" the transformation module about a reference between the pixel coordinates of the scanned map and the related coordinates for every pixel in the projected LOCATION. Ideal points would be close to the four corners of the scanned map. It is recommended to read coordinates from the map using the grid printed on the map. The related coordinates can be typed in later during the assignment. GRASS provides a tool for convenient assignment of these geographic reference points using a mouse. To do that, first start a GRASS monitor with `d.mon x0`, then the module `i.points`. It will prompt for the image group (which just contains the scanned map), in our example the group "mapscan". In the GRASS monitor, the scanned map has to be selected and then it will be displayed. In the graphical menu of `i.points`, you find a ZOOM entry. Using BOX you can enlarge the first corner of the scanned map. Make sure to zoom-in so that you can see each pixel well without losing the orientation on the map. Then, using a mouse click, select a point for which you have the coordinates from the paper map. Within the terminal window, GRASS asks you for the easting and northing of this point – type it in using the keyboard (delimited by a blank, see Figure 4.3). The same procedure has to be done for the other three corner points.

The quality of the point positioning can be directly analyzed using the ANALYZE menu entry. It calculates the "RMS error" after setting at least three reference points.[4] It should not be larger than half of the true resolution of the scanned map as we have calculated above. The overall RMS error is a sum of all partial errors (one for every matching point). If it is too large, you can delete a point from the ANALYZE table (double click to toggle a point on and off) and select a new point. Once all four points are selected and assigned properly, leave `i.points`, and the points will be saved automatically.

Finally, we perform the transformation of the scanned map using the module `i.rectify` with 1st order transformation. After starting `i.rectify`, select a 1st order polynomial (as "order of transformation"). This will perform the linear transformation (stretching and rotating). Next, enter a name for the scanned map for storage in the projected LOCATION (it may be identical). Now you have two options:

```
1. Use the current region in the target location
2. Determine the smallest region which covers the image
```

The first method is useful when you want only a subset of the scanned map (e.g. to cut off the borders). It uses the current settings of the target LOCATION. You have to be careful to preset the resolution and the current region according to the target coordinates of the scanned map. Otherwise you may obtain unwanted results. This method is useful after you get some experience.

The second method calculates the smallest region in the target LOCATION which covers the map. It may be sometimes larger than the DEFAULT_WIND definition of the target LOCATION. Here you can adjust the boundary coordinates and the desired target resolution manually. When accepting the settings, you can directly set the current region of the target LOCATION to the new settings.

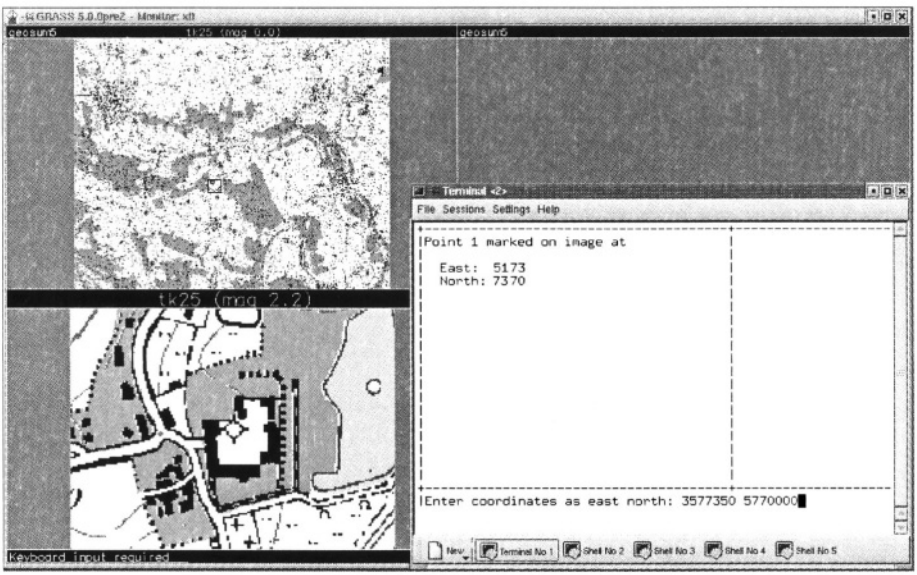


Figure 4.3. Geocoding of a scanned map with `i.points`

The module `i.rectify` then starts to transform the map. This may require some time, depending on map size, resolution, and hardware. As UNIX is capable of multitasking, you can continue working with GRASS, or even leave it, while the computation runs in the background. After the transformation has finished (at which point `i.rectify` sends an email), you can look at the new map in the target LOCATION. After restarting GRASS with the projected LOCATION and opening a GRASS monitor, the transformed map can be displayed with `d.rast`.

Quality control. The process described above takes a bit of time but leads to very accurate results when carried out with care. Quality control is always recommended, however. In combination with the zooming (`d.zoom`) the `d.what.rast` module allows us to check the coordinates of the four corner points used for the rectification. These should, of course, correlate with the equivalent points in the printed map. If the result is not satisfactory, we have to leave the target LOCATION and restart GRASS with the xy LOCATION. Now `i.points` can be called directly, because the group and target definition as well as the POINTS are still available. The accuracy can be increased by checking and improving the POINTS assignment. A new run of `i.rectify` is then necessary and the result should be checked again. The temporal xy LOCATION can be deleted after finishing the rectification as described in Section 3.1.4.

Note that this procedure is valid only for scanned, unreferenced maps. If you have digital data which are already geocoded and need to change the coordinate system, refer to Section 3.3 for an automated map transformation.

Seamless geocoding of multiple scanned maps. The transformation described in the preceding section can be used also for importing several scanned maps without gaps between boundaries. This is a way to import large maps which are too big for common scanners. The solution is to scan a large map in multiple parts. This is somewhat time consuming but useful if you do not have a large, expensive scanner.

The scanning of the map should be done with overlapping borders, to improve the identification of matching reference points. We assume that the map portions are available in a GRASS supported raster format. There also needs to be a target LOCATION, large enough to cover the complete map area.

Now set up a xy LOCATION as described above. Beware that the extent of the xy LOCATION has to cover the maximum extent of a scanned map portion. Import all map files into this xy LOCATION. Each portion will go into its own image group (so that it only contains one map) with `i.group`. Set the transformation target for all groups to the projected LOCATION with

`i.target`. Then assign coordinates for each map portion to the four map corners with `i.points` and the keyboard. Using `i.rectify` (again with a 1st order polynomial) the map portions are transformed into the projected LOCATION.

Once all scanned maps are transformed, the result can be checked in the projected LOCATION, leave the xy LOCATION for this. Now all individual maps should be checked for their correct positioning and orientation, the base accuracy. Set the current region to the maximum (i.e. the default value) with:

```
g.region -dp
```

Open a GRASS monitor and display each map by:

```
d.rast -o mapportion
```

The overlay mode (flag `-o`) allows us to overlay adjacent maps. With `d.zoom` you can now inspect whether distortions are visible between the maps. Next, cut off unwanted map borders and patch the portions to a seamless map. We explain this later on in Section 5.4.2.

4.1.5 Export

The export of raster data can be done in several ways. In GRASS 5.3 there is no general export tool available, this is planned for GRASS 5.7. Therefore a set of export modules exists which allows us to write various formats: GRASS ASCII (`r.out.ascii`), ARC/INFO ASCII GRID (`r.out.arc`), BIL (`r.out.bil`), BINARY ARRAY (`r.out.bin`), PPM (`r.out.ppm`), MPEG (`r.out.mpeg`), TIFF (`r.out.tiff`) and TARGA (`r.out.tga`).

As mentioned above, only the map portion of the current region will be exported. The export modules can be used on command line as well as interactively with menus. A few modules such as the ARC/INFO ASCII GRID, GRASS ASCII and PPM export modules optionally allow us to use UNIX piping, i.e. redirecting the data stream to another module. A piping example to produce a 8 bit GIF image is:

```
r.out.ppm elevation.dem out=- | ppmquant 256 | ppmtogif>elev.gif
```

The result of `r.out.ppm` is directly sent to `ppmquant` to quantize the 774 elevation categories to 8 bit (256 colors), then to `ppmtogif`. The data transfer is done through “standard out” (stdout) indicated by – (dash). The GIF data stream resulting from `ppmtogif` is written to the `elev.gif` file. The produced GIF file is stored into the current directory.

Export to XYZ ASCII format. A common format for raster data exchange to other GIS is the plain XYZ ASCII format (i.e. x, y coordinates with the according z value). Unlike the GRASS ASCII raster export with `r.out.ascii` (which exports the data as an ASCII matrix), the following command produces a file with one line for each cell information, each line containing three columns (easting, northing, z):

```
r.stats -1g elevation.dem nv="-9999" > altitudes.txt
```

The category label (attribute of the raster cell) is exported when using the `-1` flag, the optional `nv` parameter allows us to replace the NULL value with a different character or string.

4.2. VECTOR DATA

Line, area and point features can be represented in GRASS by vector data model. It stores the feature's geometry, attributes and topology. Three different vector types are used to store polygons (called *vector areas* in GRASS), lines (*vector lines*), and points (*vector sites*). The latter are convertible from and to the native GRASS sites data model.

4.2.1 GRASS vector data model

GRASS stores vector data using graphic elements (primitives) such as point, line, and area boundary (GRASS 5.7 stores additionally the centroid). Vector lines may consist of a single line (arc) or multiple connected lines (polylines). A closed ring of line segments defines a vector area. The end points of a vector

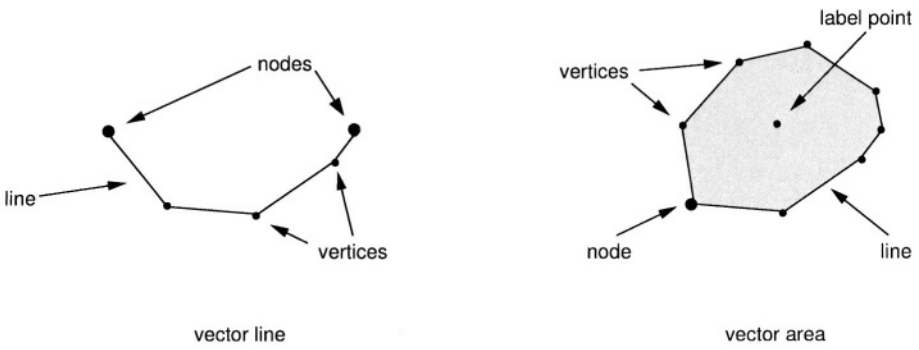


Figure 4.4. Vector types in GIS: vector line and vector area

line are called *nodes,* points along a line are *vertices* (see Figure 4.4). An area in an area is called *island.*

Vector lines and areas have attribute data assigned as a *category number* (attribute ID, also called CAT_ID) and an optional *category label* (attribute text, also called CAT_DESC). Both category number and category label may be shared with other vectors. Unlike in other systems, the internal unique vector ID is not visible to the user in GRASS 5.3. To assign attribute information to vector data, a *label point,* which links the attribute information to the geometrical data, is required. While vector lines have the label point positioned on the line, vector areas keep their label point within the area. GRASS is capable of managing only one attribute per vector internally, but a quasi infinite number when connecting the system to an external DBMS such as PostgreSQL. The internal structure of a roads map with vector ID, CAT_ID and CAT_DESC may look like this:

```
ID CAT_ID CAT_DESC
1  5      "unimproved road"
2  5      "unimproved road"
3  5      "unimproved road"
4  5      "unimproved road"
5  5      "unimproved road"
6  5      "unimproved road"
7  4      "light-duty road, improved surface"
8  5      "unimproved road"
[...]
```

This is the logical structure of GRASS vector data category numbers and labels. In the GRASS database, the internal ID (left column) is not visible to the user.

GRASS vector data model includes topology, describing spatial relations between the graphic elements that define the feature location and geometry. With this type of data structure, the common boundary between two adjacent areas is stored as a single line, and shared nodes do not have to be duplicated. With topological information, it is possible to answer the following questions from a vector data set (Bartelme, 1995:18):

- find neighborhood relationships between objects;
- analyze if one object contains another object (island areas);
- find intersections of objects;
- analyze vicinity of two objects.

Detailed rules for digitizing vector data in a topological GIS are given in the digitizing section. For discussions on general computational geometry, see the book of O'Rourke, 1998.

GRASS allows users to store the geometrical component of a vector map layer in the binary vector format, which is the default, and optionally in the ASCII vector format, mostly used for data exchange or modification by text editor. The category labels are always stored in ASCII format, and both the category numbers file and the category labels file are shared between ASCII and binary vector format.

It is generally recommended to store vector line features and vector area features in separate map layers.

GIS vector data are available in a wide range of different formats, with no single standard available, although some formats (e.g. ESRI SHAPE) are more common than others. Because of the complex data structure, exchange of vector data is often more complicated than is the case for raster data. In GRASS 5.3, the import is handled by a specific command for each different format; a vector equivalent to `r.in.gdal` is available from GRASS 5.7 onwards (`v.in.ogr`).

4.2.2 Import of vector data

After running the appropriate command, the imported vector data are stored in the GRASS binary vector format. For each vector map layer, topology has to be built. This is done either by the import module by specifying the relevant flag, or it needs to be done after the import using the module `v.support`. If the vector map does not have the topology built, commands which need the topology will print a related error message. The actual topology status can be queried with `v.info`.

Import of SHAPE files. Vector data in the commonly used ESRI SHAPE format are imported using the module `v.in.shape`. It is important to know that SHAPE files are not stored in a topological format but as "Simple Features". This may lead to problems because common area boundaries are stored duplicated (can lead to gaps and slivers problems). The module `v.in.shape` contains an internal "topology engine" which fixes a lot of common SHAPE file problems. The parameters `snapdist` and `sliver` are useful when importing a manually digitized SHAPE map to properly snap common lines. Files can be read from any directory; if no path is specified, the current directory is used. Because GRASS 5.3 internally supports only a single attribute per vector, you have to select the column with the attribute that you want to store while importing. When using the flag `-l`, only the table field definitions (field names and types) of the associated `.dbf` file are listed. For the data import, the columns used for category number and category label have to be selected from the DBF table:

```
v.in.shape -l roads.shp out=myroads
v.in.shape roads.shp out=myroads scale=1:24000 att=CAT_ID\
                                              label=CAT_DESC
```

Because the map scale is not stored in the SHAPE files, the appropriate scale should be specified during import. Please refer to the module manual page for details.

Import of E00 files. Vector data in the ESRI E00 format are imported with `m.in.e00`. The E00 format is preferred to the SHAPE files because it keeps a better data structure as well as the projection information. The attribute import differs from `v.in.shape` in that `m.in.e00` imports all attribute columns and stores them in different category label files while generating a single map containing the vector geometry. To illustrate how to create vector map layers with the desired attributes we import the vector map:

```
m.in.e00 -s roads.e00
```

The name of the imported file in GRASS will be `roads` and, with the flag `-s`, the topology is automatically built. Assume that this E00 file contains the columns with attributes called "width" and "velocity". The best way to create the two vector map layers (one for each attribute) is to generate copies of the vector geometry map with the same names as the category label files. This leads to replicated geometry files but is the only way to make all attributes available without external DBMS. For this modification you can look into the GRASS DATABASE to get the names of the category label files. The procedure is somewhat inconventient, a much smarter attibute management is provided with GRASS 5.7. First we change into the GRASS DATABASE into the MAPSET subdirectory. The `g.gisenv` command delivers the required variable names:

```
cd `g.gisenv GISDBASE`
cd `g.gisenv LOCATION_NAME`/`g.gisenv MAPSET`
cd dig_cats/
ls -ltr
```

The flag `-l` generates a long (complete) directory listing; the flag `-t` sorts by timestamp and `-r` reverses the sort order to have the newest files at bottom for convenience. By running this command you will see the names of the recently generated category label files. In our example there will be the files `roads.width` and `roads.velocity` according to the attribute columns in the E00 file. We have to preserve the category label files from overwriting during this procedure, so we change their names to temporary names:

```
mv roads.width roads.width.tmp
mv roads.velocity roads.velocity.tmp
```

Now copy the vector geometry map to the desired new names:

```
g.copy vect=roads,roads.width
mv roads.width.tmp roads.width
v.info roads.width

g.copy vect=roads,roads.velocity
mv roads.velocity.tmp roads.velocity
v.info roads.velocity
```

The `v.info` command should report that both vectors and attributes are stored in the map. Generally this procedure can be performed for all category label files in the `dig_cats/` directory which were generated by `m.in.e00` during a E00 file import.

The module can read E00 files from any directory. If no path is specified, the current directory is used.

Import of UNGENERATE files. The ESRI UNGENERATE format is a generic ASCII vector exchange format which is supported by numerous GIS. Depending on the vector map type (line vectors or polygon vectors) it consists of a differing number of files. Line vector maps, also called "line coverages", are two files with `.lin` (vector lines) and `.dat` (line attributes) extension. Polygon vector maps, also called "polygon coverages", consist of three files with `.pol` (vector polygons), `.pnt` (attribute label points) and `.dat` (polygon attributes) extensions. You have to take care to receive or provide a complete set of files when exchanging data in UNGENERATE format.

These data are imported with `v.in.arc`. An import example for a polygon map:

```
v.in.arc type=polygon lines_in=topo12.pol points_in=topo12.pnt\
text_in=topo12.dat vector_out=topo12 idcol=1 attcol=2 catcol=4
```

It is important to specify the correct columns for the parameters `idcol`, `catcol` and `attcol`. The parameter `idcol` links to a numerical field containing the vector IDs (line-IDs). The parameter `attcol` links to the category numbers (attribute IDs) which also contains numbers. The parameter `catcol` links to the category labels which may be an attribute text label. Files can be read from any directory. If no path is specified, the current directory is used. An import example for a line map:

```
v.in.arc type=line lines_in=topo12.lin text_in=topo12.dat\
               vector_out=topo12 idcol=1 attcol=2 catcol=3
```

After import the topology has to be built by:

```
v.support topo12
```

Generally, we recommended setting the map scale to the true map scale. As the scale is unfortunately not stored in the UNGENERATE format (GRASS will set an imported UNGENERATE map to 1:1 scale per default), the map scale can be set with the module `v.digit`. The first metadata screen after loading the vector map (for "Select digitizer" select "none") provides the entry for the map scale. After leaving this screen with <ESC><ENTER>, `v.digit` can be left: Either answer "Shall we continue? [y]" with `n` or leave it from the main menu with `Q` (quit).

Import of GRASS ASCII vector files. The generic GRASS ASCII vector format is very similar to the UNGENERATE format. For lines, the label points are positioned on the vector lines. For polygons, the label points are stored within the polygons. If these rules are not fulfilled, `v.support` will print a warning ("PNT_TO_AREA failed"). If you have an imported binary vector map with some topological problems, exporting the map to ASCII format is a way to investigate the data. Each line or polygon is written into an ASCII file which can be edited with a text editor. This file has an associated label points file and an attribute table.

The module `v.in.ascii` expects the vector lines file within the LOCATION and MAPSET in subdirectory `dig_ascii/`. The category labels (attributes) have to be stored in `dig_cats/` (in this directory, GRASS keeps its category label table as ASCII files). The label points file is written into the directory `dig_att/`. To get some experience, it is a good idea to export an existing vector map layer and study the resulting files – for an example, see the paragraph "Export into GRASS ASCII vector format" in the next Section.

The ASCII vector map can be converted to a GRASS binary vector map with `v.in.ascii`. As mentioned, the module will read the associated files from the `dig_ascii/`, `dig_att/` and `dig_cats/` directory. The resulting GRASS binary vector file is stored in `dig/` while the other files remain unchanged.

Import of SDTS files. SDTS (Spatial Data Transfer Standard) and the TVP (Topological Vector Profile) define two basic types of spatial objects: simple spatial objects, i.e., lines, polygons, nodes, etc. and composite objects, which are made up of one or more other simple and/or composite spatial objects. SDTS composite objects, which GRASS cannot handle directly, are imported as records in DBMS-ready tables.

`v.in.sdts` creates one or more GRASS vector maps in the current MAPSET from a Spatial Data Transfer Standard data set conforming to the Topological Vector Profile (TVP). The program generates file within the LOCATION in the directories `dig/`, `dig_att/`, and `dig_cats/`. If requested, files of attributes in database-ready form are created, along with scripts to create an appropriate

SQL-compliant relational database and load the attribute files into the new database. Special database-ready files of tables linking the attributes to the GRASS vector map layer or layers are also generated. The source SDTS data set must be in the user's current directory. Before importing, it is possible to look at the file's contents with flag `-i`:

```
v.in.sdts -i catd=ROADCATD.DDF
v.in.sdts catd=ROADCATD.DDF output=sdtsroads
```

We can use the module `m.sdts.read` for querying info about SDTS files, such as the data quality, lineage, etc. It reads SDTS or other data from files in ISO 8211 (FIPS 123) format and dumps contents to screen and/or file:

```
m.sdts.read in=ROADCATD.DDF
m.sdts.read -s in=FFFFCATD.DDF out=sdtsdump
```

With first command the data are shown record-wise, the second command dumps the data into text file `sdtsdump`. For further details please refer to the manual pages.

Import of DXF files. GRASS supports import and export of vector maps in *DXF format.* Both 2D and 3D DXF are supported for reading, but GRASS 5.3 stores the vector information as 2D vectors with an attribute.

As an example we show the import of a 3D DXF contour line map. This requires a few steps. When only specifying the map name and no further parameters, the DXF map is completely imported. In case of different map layers existing within the DXF file, these layers will be written into separate GRASS vector maps. Alternatively, you can specify the `line` selector to only import a subset of layers or just a single layer. As an example, we import the layer "CURVES" from the DXF file `contour.dxf` which is stored in the current directory. The first command `v.in.dxf` will print a warning "WARNING: 3-d data in dxf file" in case 3D data such as contour lines are present. This indicates that `v.in.dxf3d` should be run afterward to attach the elevation information to the line vectors (not required for 2D DXF files):

```
v.in.dxf dxf=contour.dxf line=CURVES
v.in.dxf3d dxf=contour.dxf line=CURVES
v.support contour.CURVES
d.vect contour.CURVES
d.what.vect
```

The GRASS vector name is a composition of the imported DXF file name and the selected layer. The layer names, as stored in the DXF layer, are displayed at the end of the import process. For an inspection, the imported map may be displayed with `d.vect` and queried. The query module `d.what.vect`

allows us to select vectors by mouse and to print their attributes. Such vector queries are explained in Section 6.3.1 in further detail.

Note that, apart from the GRASS binary format, DXF files can be imported also into GRASS ASCII format using the flag `-a` (`v.in.dxf`). This will require a subsequent run of `v.in.ascii` to convert the map to a GRASS BINARY vector map for the later usage with other vector modules. Please refer to the manual page for further details. If desired, you can also specify the optional parameter `prefix` which allows us to define the name of the map within GRASS. The prefix is extended with the layer name extension.

When importing DXF polygons, a problem occurs because the second mentioned module `v.in.dxf3d` works correctly only for line vectors. As a workaround you can run `v.line2area` after importing the DXF file to convert the vector type from lines to polygons. Subsequently use `v.alabel` to label the polygon vectors. Finally, use `v.support` to build the topology.

When importing DXF contour maps, it is possible to import a 3D contour DXF file in one step with the script `v.in.dxf3d.sh`. It internally runs the `v.in.dxf`, `v.in.dxf3d` and `v.support` modules. It requires to specify the DXF layer names for the layer keeping the contour information and the layer for the "z" values (the name may be identical):

```
v.in.dxf3d.sh dxf=contour.dxf line1=CURVES line2=CURVES
```

Coordinate transformation for xy DXF vector data. Often DXF data are delivered in non-georeferenced xy coordinates. To use them along with other GIS data, these coordinates have to be transformed to the coordinate system of the current LOCATION. The module `v.transform` requires a table of ground control points (GCPs, also called tie points) to perform this transformation. It is a table of points with xy coordinates and their corresponding georeferenced coordinates. To generate this table, coordinates of points such as road intersections etc. are identified in the DXF map and another corresponding reference map. Also, GPS points measurements can be used.

In GRASS, the DXF map geocoding process is three-fold. First, the DXF data are imported into the projected LOCATION, although the DXF map keeps its xy coordinates reference. Second, the ground control points are identified within the imported DXF map as well as the reference map or taken from GPS measurements and stored in a text GCPs table. Finally, the imported map is transformed to the current LOCATION coordinate system by shifting and rotating the input DXF map using the GCPs table.

To illustrate the procedure we will import the DXF map `vmap.dxf` (xy coordinates) and transform it to the map `refmap` stored in a georeferenced LOCATION, in this case in the Gauss-Boaga Grid System (Italian national grid system). To start, the DXF map is imported into this LOCATION as a GRASS ASCII vector map:

```
v.in.dxf -a dxf=vmap.dxf prefix=vmap
```

For every DXF layer (DXF lines, DXF labels, ...), an ASCII vector map is created within the LOCATION in the subdirectory `dig_ascii/`. In our example it will be an ASCII vector map `vmap.lines`. To build its topology, this map then needs to be converted to the GRASS binary vector map `vmap_xy`:

```
v.in.ascii in=vmap.lines out=vmap_xy
v.support vmap_xy
```

Now the DXF map is in the xy coordinate system in the LOCATION with Gauss-Boaga Grid System. Next step is to identify the ground control points by selecting them from the maps displayed in the GRASS monitor. To do that, we have to reset the current region to the imported DXF map before using the display command, otherwise we would see nothing (i.e., the map is located outside the displayed area). The adjustment of the current region is done with `g.region` by specifying the vector map name. The optional flag `-p` will also output the map boundary coordinates:

```
g.region -p vect=vmap_xy
d.erase
d.vect vmap_xy
```

The command `d.erase` sends the updated coordinates to the GRASS monitor and erases the screen, and then `d.vect` displays the map.

Because we do not expect internal map distortions, we have to identify only four ground control points for the transformation. To get these GCP xy coordinates for the imported DXF map, we can use `d.where`. We store the coordinates of these four GCPs of the xy vector map in an ASCII file (e.g. called `POINTS`) in the `$HOME` directory. We can use any text editor to write each xy coordinate pair on a separate line. The next step is to find the corresponding four GCPs in the projected reference map or to assign corresponding coordinates from a GPS measurement. When using GPS data, just enter the georeferenced coordinates values. When using a reference map in the current LOCATION, for example the vector map `refmap`, the current region has to be reset to this map and the map should be displayed:

```
g.region -p vect=refmap
d.erase
d.vect refmap
```

Again, `d.where` helps to get the corresponding four GCP coordinate pairs from the reference map. Store these values in the ASCII table next to the related xy coordinates as EASTING NORTHING with all values delimited by a space. Such a table may look as follows (the first two values are the X and Y coordinates, the second two values are EASTING and NORTHING):

```
838.8  244.2 1661701.3 5108172.4
796.2  192.6 1661643.2 5108204.3
796.2   91.6 1661543.7 5108181.8
1110.6 379.0 1661864.4 5108076.6
```

Now the coordinate transformation from xy coordinates to Gauss-Boaga coordinates can be performed. The module `v.transform` will read the GRASS ASCII vector file `vmap.lines` which is stored within the LOCATION in the related MAPSET subdirectory `dig_ascii/`. The command sequence for our example is:

```
v.transform in=vmap.lines out=vmap_trans point=$HOME/POINTS
v.in.ascii in=vmap_trans out=vmap_trans
v.support -r vmap_trans
```

The module transforms the coordinates of the nodes and vertices of the DXF map and prints out the RMS (root mean square) error for each GCP. An acceptable RMS error for a 1:24,000 map would be 1.2 to 2.4 meters. If this is not reached, the selection of the GCPs should be redone. Finally, the transformed map must be converted to the binary vector format with `v.in.ascii` for later usage (same names are allowed) and the topology generated.

To verify the transformation result, we have to reset the current region to the transformed map. Then we can display it over the reference map:

```
g.region -p vect=vmap_trans
d.vect refmap
d.vect vmap_trans col=red
```

Both maps should match well.

Import of GSHHS Shoreline files. An interesting vector data set with excellent accuracy is the GSHHS (Global Self-consistent, Hierarchical, High-resolution Shoreline Database, Wessel and Smith, 1996). The sources of data were the CIA World Data Bank II, designed for a scale of about 1:2,000,000, and the NOAA World Vector Shoreline, which is at a scale of 1:250,000. The GSHHS shorelines are constructed entirely from hierarchically arranged closed polygons. The data can be used to simplify data searches and data selections, or to study the statistical characteristics of shorelines and land-masses. The data set can be accessed from the GSHHS Web site[5] and from the National Geophysical Data Center (NGDC), Boulder, Colorado (USA). The related GRASS module is `v.in.gshhs`. It optionally allows us to geographically limit the import to a subregion. The GSHHS data are freely available at five different resolutions: crude, low, intermediate, high, and full. The [c,l,i,h]-versions were derived from the GSHHS full resolution file using the Douglas-Peucker algorithm which simplifies a line segment given a tolerance (0.2 km

to 25 km). GSHHS coordinates are originally in the latitude-longitude coordinate system. If required, they can be transformed during import to the current LOCATION coordinate system. A selected subarea can be imported by specifying boundary coordinates, or by using the flag `-g` to use the current GRASS region as boundary constraints. The flag `-a` allows us to import vector data as area type (default is line type). The following example imports an intermediate resolution shoreline map file into GRASS using the current GRASS region and automatically runs `v.support` after import:

```
v.in.gshhs -gs gshhs_i.b out=shoreline.lowres
```

This imports the map into the current region, clipping it to the region boundaries and builds the topology.

4.2.3 Export of vector data

Data exchange of GRASS vector data with other GIS is supported in various formats.

Export into SHAPE format. To export GRASS vector files to ESRI SHAPE format, the module `v.out.shape` can be used. A GRASS vector file to be exported to SHAPE may contain both vector lines and vector polygons. If you select the parameter `type=both` for conversion, two layers will be produced with the appended suffixes `_line` and `_area` respectively for lines and polygons. However, lines used as area edges are not duplicated in the lines layer. Please keep in mind that it is generally recommended to store vector lines and areas in separate map layers. The `cats` parameter allows us to optionally export category labels if they are present. They may be written according to their type as character string, integer, or float number into the associated DBF file. The flag `-v` enables verbose output to see the export progress.

As an example, we export the vector roads map of the Spearfish data set, stored in MAPSET PERMANENT. The stored vector types can be checked with `v.info` which reports whether the topology is present, the numbers of category labels, vector lines, areas and islands (areas in areas). To export, the vector type has to be set accordingly ("line", "area" or "both"). An example for the Spearfish `roads` map which contains only lines is:

```
v.info roads
v.out.shape -v roads type=line mapset=PERMANENT cats=string
```

Three files are written into the current directory: `roads.shx`, `roads.shp` and `roads.dbf`. From GRASS 5.7 onwards also the corresponding `.prj` file containing the projection information will be generated when exporting to SHAPE format. The written DBF file may be inspected with `gnumeric`, a free spreadsheet software.

Export into E00 format. A recommended format for exporting GRASS vector data is the ESRI E00 format. It is performed by `v.out.e00`, which writes a vector map to an ARC/INFO line/polygon coverage. We can export the `roads` map of the Spearfish data set as follows:

```
v.out.e00 roads mapset=PERMANENT out=roads.e00
```

It is required to specify an output name, otherwise the E00 data stream will be written to "standard out" (stdout). The file is stored in the current directory.

Export into UNGENERATE format. Export of GRASS vector maps to ESRI UNGENERATE file format is possible with `v.out.arc`. A GRASS vector file to be exported to ARC/INFO must either contain only linear features (line vectors) or only area features (polygon vectors). The vector type, line or polygon has to be specified. To inspect a vector file, use `v.info`. As an example we export the `roads` vector map by:

```
v.info roads
v.out.arc roads type=line arc_prefix=roads
```

Export into GRASS ASCII vector format. The module `v.out.ascii` allows us to export a GRASS vector map to GRASS ASCII format. As an intended exception, the exported file will be stored within the LOCATION in a subdirectory `dig_ascii/` and not in the current directory. The category labels are stored in ASCII format at `dig_cats/` (however, the file is identical to the binary GRASS binary format). The label points file will be found in the directory `dig_att/` which is also identical to the GRASS binary format.

Export into SDTS format. To export a vector map to SDTS data set conforming to the Topological Vector Profile (TVP) we can run the module `v.out.sdts`. The usage is very simple:

```
v.out.sdts roads output=ROAD
```

The parameter `output` is a prefix for the SDTS output files which must be 4 upper-case characters and/or digits. The resulting files are stored in the subdirectory named according to the `output` parameter, generated in the current directory or a directory defined with `path` parameter. The output data set is in the mandatory ISO 8211 (FIPS 123) format. When desired, the ISO 8211/SDTS output files can be inspected with `m.sdts.read`. Two flags are available: `-a` restricts the output to areas while omitting lines, the flag `-m` accesses a user-defined metadata file. This file is typically, but not necessarily, created with `v.sdts.meta`. SDTS data sets are required to contain 5 different data quality report modules, for lineage, positional accuracy, attribute accuracy, logical consistency, and completeness. When `v.out.sdts` is run, it searches in the

user's MAPSET `dig_misc` directory for appropriate files, one for each module, containing narrative text in ASCII format as written by `v.sdts.meta`, a TclTk based application with menu entries to supply the metadata information. If metadata files generated with `v.sdts.meta` is found, they are converted to SDTS/ISO 8211 format and added to the export data set; warning messages are displayed if any data quality modules are missing.

In addition to the files created by `v.out.sdts`, every SDTS transfer must contain a README file. This file is not generated by `v.out.sdts`, and must be created by hand (see the manual page for this command).

Export into DXF format. To export GRASS vector files to DXF format, the module `v.out.dxf` can be used. The usage is very simple, as we demonstrate with the roads map again:

```
v.out.dxf roads out=roads.dxf
```

The DXF file is written to the current directory.

4.3. SITES DATA

In addition to the vector format, features that are spatially referenced to a single point can also be stored in the GRASS site format. Note that other GIS software may not support handling of site data separately from vector data.

4.3.1 GRASS sites data model

The multi-dimensional, multi-attribute ASCII sites format includes a coordinate entry followed by a category (with a # as prefix), optional floating point attributes (% as prefix), and optional character string labels (@ as prefix). All character strings that include a space must be enclosed with quotes " ". The coordinates are separated by a vertical bar | while the rest of the fields are separated by a space. The format in general is as follows:

```
east|north|[z[|...]][#cat_int] [%at_fp [%at_fp...]]\
                               [@at_str [%at_str...]]
```

The GRASS sites model also supports an optional header which describes the data stored in the sites file. Example of such a header is as follows:

```
name|myname
desc|Soil chemical data
time|15 Jan 1994
labels|east north elevation ID pH Nitrogen
form|||#%%
```

The following example of soil profile data entry includes location coordinates easting and northing, site ID as an integer category, pH value as a floating point attribute, and character strings describing the soil texture and land cover. In our example, the file header includes only two lines describing the name of the file and the label for each field:

```
name|Soil data
label|east north ID pH texture vegetation
3567000|5787000|#1 %4.2 @"sandy soil" @"forest"
```

For points in geographical coordinates, the header and data entry may look like this:

```
labels|longitude latitude ID elevation
75:33:55.34W|33:44:59.21N|#1 %225.2
```

4.3.2 Import of sites data

There are several approaches to import a sites map layer. Site data can be directly entered into the GRASS DATABASE by copying an ASCII file in the prescribed format into the `site_lists/` subdirectory in the LOCATION/MAPSET directory, or the data can be imported from a number of supported formats (e.g., CSV, ArcView, dBase, MapInfo). Finally, the points can be digitized from a map.

GRASS provides modules to import point data in various formats such as `s.in.mif` for MapInfo point data, `s.in.shape` for ArcView, `s.in.atkisdgm` for German ATKIS (elevation data transfer format), and `s.in.dbf` for the dBase data. Site data measured by GPS can be imported as sites using `s.in.garmin.sh` or as vector data using `v.in.garmin.sh` and then converted to sites using `v.to.sites`. Timestamps or time ranges are supported with a `date` parameter for `s.in.ascii` and `s.in.dbf`. Please refer to Section 5.1.4 how to define a timestamp or time range.

Importing ASCII tables. Point data in ASCII format can be imported directly with `s.in.ascii`. The data should be arranged as `east north value`:

```
3246000.0 5877000.0 23.0
3246012.5 5877012.5 24.4
3246025.0 5877025.0 22.3
```

Then, they can be imported as follows:

```
s.in.ascii sites=mygrasssites input=myasciisites
```

Optionally, you can define a timestamp for the map (parameter `date`). Note that `s.in.ascii` will insert a file header and a category number (ID) after

your east and north coordinates and import your value as a floating point attribute, so your data in GRASS sites format will look like this:

```
name|mygrasssites
desc|s.in.ascii sites=mygrasssites input=myasciisites fs=space
3246000|5877000|#1 %23
3246012.5|5877012.5|#2 %24.4
3246025|5877025|#3 %22.3
```

If the input point data include a third dimension (depth, elevation) and the values are separated by commas (CSV format), you can import them with `s.in.ascii`, using the parameter `d` for setting the number of dimensions and `fs` for a delimiter as follows:

```
s.in.ascii sites=grasssites3d input=myasciisites3d d=3 fs=,
```

Depending on application, elevation `z` can be handled either as a third dimension `east|north|z|` or as a floating point attribute `east|north|#id %z`.

As we have shown in the Section 5.1.6, UNIX provides an effective tool for working with ASCII data called `awk`. You can use it to modify your ASCII data to conform to the GRASS site format. For example, if your data in `points.dat` are organized as:

```
ID north east elevation
```

you can remove the ID and change the order of coordinates as required by GRASS using:

```
cat points.dat | awk '{print $3, $2, $4}' | s.in.ascii mysites
```

which writes the values from the third (east), second (north) and fourth (elevation) column and then imports this file by `s.in.ascii`. You will find more detailed explanation of `awk` and related examples in Appendix A which describes the use of UNIX text editing tools.

Importing site data in geographical coordinates. Latitude-longitude data can be imported either in DD format (decimal degree, see Section 2.2.2) with longitude given in the first and latitude in the second column:

```
8.314824 54.921730 site1
8.897605 54.872353 site2
9.549371 54.834080 site3
```

or in DMS (degree, minutes, seconds) format:

```
8:18:53.3664E 54:55:18.228N site1
8:53:51.378E  54:52:20.471N site2
8:32:57.7356E 54:50:02.688N site3
```

The module `s.in.ascii` will accept point data written in any of these two formats.

4.3.3 Export of sites data

Site data can be exported to an external ASCII file by `s.out.ascii`. The `s.out.e00` program is designed to create an ARC/INFO E00 ASCII points file. Alternatively, point data can be converted to raster data by `s.to.rast` and exported to ARC/INFO ASCII GRID by `r.out.arc` or other GIS raster formats.

NOTES

1 GDAL library, `http://www.remotesensing.org/gdal/`

2 Spearfish 10 m LatLong/NAD83 DEM ArcGRID coverage file, section data sets, `http://mpa.itc.it/grasstutor/`

3 Netpbm tools, `http://sourceforge.net/projects/netpbm/`

4 The "RMS error" represents the distance of the set matching point towards the ideal placed matching point. It is calculated through:
$rms = \sqrt{(x - x_{orig})^2 + (y - y_{orig})^2}$

5 GSHHS shoreline data Web site,
`http://www.soest.hawaii.edu/wessel/gshhs/gshhs.html`

Chapter 5

WORKING WITH RASTER DATA

In this chapter we explain the processing of raster data within GRASS, including some examples of spatial analysis. The raster data model, especially when combined with map algebra, provides wide range of capabilities for spatial modeling, all of which would be impossible to cover within a single chapter. Therefore, this chapter provides the basic description of the tools; more can be learned from the manual pages, tutorials and publications provided on the GRASS Web site.

For a description of the GRASS raster data model, as well as raster data import and export, please refer to Section 4.1.3.

5.1. VIEWING AND MANAGING RASTER MAP LAYERS

In this section we continue to use the Spearfish data set to illustrate our examples. Please refer to Section 3.1.3 on how to start GRASS with the Spearfish LOCATION.

5.1.1 Displaying raster data and assigning a color table

Raster map layers can be displayed using the `d.rast` module. First open the GRASS monitor, then run the display module:

```
d.mon x0
d.rast elevation.dem
```

The map will be displayed in the GRASS monitor. The module `d.rast` offers two useful optional parameters, `catlist` and `vallist`. When us-

ing `catlist` you can selectively display categories for integer map, while `vallist` applies to floating point maps. For example,

```
d.rast geology catlist=4,5 bg=black
```

shows only categories 4 and 5 of the `geology` map (i.e. sandstone and limestone for the Spearfish area) with black color for the other raster cells. You can run

```
d.rast aspect vallist=225-315 bg=blue
```

to see southern exposed slopes (note that in GRASS the aspect angles are calculated from east counterclockwise). The GRASS monitor can be erased with `d.erase`.

A nice script is `slide.show.sh` which just requires an open GRASS monitor. When running it, it will show all available raster maps. Optionally you can define a name prefix to see only selected maps. To learn how to view raster data in 3D using the `nviz` module, see Chapter 8.

Color tables. Each raster map has its own color table. When no color table file is present, the rainbow color coding will be used (which may not be satisfying in many cases). Colors can be assigned with `r.colors`. The module provides a range of pre-defined color tables. To give it a try, change the color table of the aspect map to `waves`:

```
d.rast aspect
r.colors help
r.colors aspect col=wave
d.rast aspect
```

As the chosen color coding may be unusual for an aspect map, we can restore the typical color coding for aspect maps by:

```
r.colors aspect col=aspect
```

Later on, in Section 5.4.4, we will show how to fine-tune a color table using user defined color values and you will find a number of examples for creating color tables in Chapter 12.

Displaying a legend and a scale. The legends for raster maps can be displayed using the module `d.legend`:

```
d.rast fields
d.legend fields
```

If you want to define the location and extent of the legend using the mouse, include the `-m` flag. Then click the left mouse button to "Establish a corner" and

subsequently the right button to "Accept box for legend". The menus for these interactive display commands are always explained in the terminal window. For continuous field data such as elevation models, the `-s` flag can be used to draw a continuous color gradient legend. If you have selectively displayed only certain categories or values using the `catlist` or `vallist` options for the `d.rast` command, you can selectively display the relevant colors in your legend using the `use` option for a list of categories and the `range` option for a range the map values.

You can place a map scale with `d.barscale`, choosing its location using mouse.

5.1.2 Raster map queries and profiles

One of the most common tasks when working with digital maps is querying the values at a given location. This can be done either interactively using a mouse, or on the command line for a location given by its coordinates. To query a single or multiple map layers interactively use `d.what.rast`:

```
d.rast geology
d.what.rast geology
d.what.rast texture,geology
```

By clicking at a certain location, you will get the coordinates at that point as well as the category values and labels in the given map layer(s). The command `d.what.rast` will also work without specifying a map name when a raster map is displayed in the GRASS monitor. When you are finished with the query, don't forget to end the request mode using the right mouse button within the GRASS monitor. To get only the coordinates at a point by mouse click, use `d.where`.

To generate a profile, you can run `d.profile`. It allows you to interactively draw profiles within the GRASS monitor. You may try it with the elevation map:

```
d.profile elevation.dem
```

Then follow the menu provided within the monitor.

Non-interactive query can be performed either at individual points or at a transect (profile along a line). With module `r.what` you can query the map at a point given by its coordinates:

```
r.what -f geology east_north=598514,4921908
```

With `-f` flag, the category label is included in the output.

To generate a profile with parameters provided on the command line, you can use `r.profile`. It outputs the raster map values located on user-defined

line(s). You can write the result to a file or to standard output (stdout). For example, we want to get a profile along a line specified by coordinates of two GPS tracking points:

```
r.profile geology out=- profile=595346,4921504,595518,4915456
```

The output list has two columns. The first column shows the cumulative transect length (cum_length), the second column contains the category value found at the corresponding location. If you additionally specify the flag -g, you get the output: easting, northing, cum_length, map_value. Alternatively, you can use `r.transect` for profiles defined by a starting point, direction and distance.

The module `r.report` can be used to report statistics for raster maps. For the given raster map, it will write out a table containing category numbers (cell values), category labels, and optionally area sizes in units selected by the user for the parameter `units` (see examples of the output in Chapter 12). Unlike the overall statistics calculated by `r.report`, the command `r.stats` operates cell-wise. It outputs individual cell values (left to right, top to bottom) in a format that can be customized using flags. Often, the result is piped to other tools such as `awk` for further processing (see an example in Section 5.1.6).

5.1.3 Zooming and generating subsets from raster maps

You can interactively zoom into a selected location within a map displayed in the GRASS monitor using the command:

```
d.zoom
```

Use the left mouse button to define the first corner point, then move around the mouse to open the zoom box and use the middle mouse button to set the second zoom box corner. If the zoom level is acceptable, confirm it with right mouse-click. The related mouse button menus are explained in the terminal window. To zoom out with `d.zoom`, use the middle mouse button. You can save the region defined interactively by `d.zoom` using the `g.region` command. The GRASS monitor can be erased with `d.erase`.

Besides zooming, `g.region` can be used to adjust the current region settings to well defined region boundary values. Note that you have to run `d.redraw` (or the sequence `d.erase;d.rast map`) after using `g.region` as the GRASS monitor needs to get the information about the changed current region. The `d.erase` which is internally used by the `d.redraw` script sends the updated coordinates to the monitor while erasing its contents. Therefore, a redraw is required to get the map(s) back.

To get a list of the maps currently displayed in the GRASS monitor, run:

```
d.frame -l
```

Generating map subsets. If you have a large raster map, but you want to work only on a small subset of that area, the subset of interest can be selected and stored into a separate map. This will save you processing time, especially when you want to try a more complex calculation before applying it to the full map.

GRASS raster computations are always limited to the current region at current resolution. After defining the area of interest with `g.region` or `d.zoom`, you can use the module `r.mapcalc` to extract the map portion into a new map:

```
r.mapcalc "newmap=oldmap"
```

Through this simple expression the map portion defined by the current region is saved as `newmap`, copying the cell values from the larger original map `oldmap`. As an example, let us copy the residential area in the north-west of the Spearfish area from the SPOT satellite image. We can use `g.region` to first adjust the current region settings (map boundaries and resolution) to a map, in this case the satellite image `spot.image`. After adjusting the region (you need to have open a GRASS monitor, if not, run `d.mon x0`) we display the SPOT image. Then we zoom into the residential area in the north-west of the satellite image. You will see the highway and, in blue shades, the residential area with the dense street network. This `spot.image` is a false color composite, so the colors are unnatural. For now, it is sufficient if you roughly zoom into the residential area. When zooming, always follow the mouse menu settings explained in the terminal window. Now we can extract the residential map through copying it from the base image (`d.erase` is required to send the new boundary coordinates to the GRASS monitor):

```
g.region rast=spot.image
d.erase
d.rast spot.image
d.zoom
r.mapcalc "residential=spot.image"
```

The new raster map `residential` contains only the zoomed subset of `spot.image`. You can look at it by running:

```
d.erase
d.rast residential
```

When zooming out (`d.zoom`, middle mouse button, right to quit), you will realize that the image `residential` contains only the desired portion of the source, this area is surrounded by no-data (NULL) raster cells.

5.1.4 Managing metadata of raster maps

Information about the data source, accuracy, producer, date of mapping or image acquisition, date of map production, and eventual modifications, is called metadata ("data describing data"). Data documentation is crucial for GIS work, for evaluation of data quality and suitability for a given task. This is particularly important for long-term projects, or where GIS data are shared with other users.

GRASS offers the option of maintaining a "history file" for documentation of a map. Many analytical modules save their calculation steps into the history file automatically. But it may be necessary to store additional information. The "map history" can be modified using the command:

```
r.support
```

in interactive mode, i.e. starting it without any parameter. After entering the map name you may proceed with <ENTER> to the question: "Edit the history file for [map]?". Confirming with `y` an input screen showing the metadata for this map opens up. Especially the fields for "Data source" and "Data Description" should be filled in. For example, we may want to edit the metadata for the recently created map `residential` (see above). Start `r.support`, select the map `residential` and proceed to the "history" screen. You can see that map date, title, creator, a description containing the map creating method "generated by r.mapcalc" and a few more entries are already stored there. You may fill the field "data source" with "SPOT 1 MSS false color composite 5/27/89". With <ESC><ENTER> you reach the next screen where you may apply further comments. Another <ESC><ENTER> takes you back to the questionnaire mode of `r.support`; you can skip the rest of the questions with <ENTER> and leave the module.

To display the metadata of a raster map, use:

```
r.info residential
```

This will display the data description, boundary coordinates, and data range. If desired, you can even email this information to yourself or to someone else:

```
r.info residential | mail -s "residential map" recip@domain.org
```

Raster map timestamps. Because a lot of mapping and monitoring produces time series of spatial data, it is important to store the relevant temporal information. GRASS allows us to store it separately from the history file using the module `r.timestamp`. This command has two modes of operation. If no date argument is supplied, then the current timestamp for the raster map is printed. If a date argument is specified, then the timestamp for the raster map is set to the specified date(s). An example for an absolute timestamp:

```
r.timestamp residential date="27 may 1989 17:58:48 -0700"
```

Another call to

```
r.timestamp residential
```

will query and show the defined timestamp. When specifying two comma-separated timestamps (e.g. `date="27 may 1989, 28 may 1989"`), they are treated as time range. Also relative timestamps can be specified:

```
r.timestamp rastermap1 date="15 hours 25 minutes 35.34 seconds"
r.timestamp rastermap2 date="100 days"
```

Timestamps can be removed by:

```
r.timestamp residential date=none
r.timestamp residential
```

These timestamp definition rules also apply for related vector and sites modules.

5.1.5 Reclassification of raster maps

Reclassification of a raster map creates a new map based on the transformation of existing categories in the original map to a new set of categories. Usually, ranges of categories are grouped into a new class using the module `r.reclass`. Those category numbers which are not explicitly reclassified to a new value will be reclassified to NULL. Before using `r.reclass` you need to know:

- transformation rules (reclass table) describing which old categories will be assigned into which new categories;
- optionally, names for the new categories (category labels).

We recommend using the module on the command line and storing the reclass table in a file. This is convenient in the case that additional modification of the reclass table is required. The file containing the reclass rules is read from standard input (i.e., from the keyboard, redirected from a file, or piped through another program). The following examples illustrate the concept.

First, we will reclassify the raster map `roads`, which includes five categories (you may check that with `r.report roads`). The new map will include class 1 (good quality) which will be assigned to the categories 1, 2 and 3 in the input raster, and the class 2 (poor quality) which will include the categories 4 and 5. We store the required reclass rules into a text file `roads.recl`:

```
1 2 3 = 1 good quality
4 5   = 2 poor quality
```

To apply the rules to the roads map and create a new reclassed map `roads.qual`, we run:

```
cat roads.recl
cat roads.recl | r.reclass roads out=roads.qual\
                           title="Road quality"
```

The first command just shows the table. We send the table to the GRASS module using the pipe into the second command. With `d.rast` or `r.report` you can check the resulting map `roads.qual`.

The second example explains reclassification of a continuous field map. We want to reclass the elevation map into elevation ranges reflecting the typical vegetation. To cover all elevations appearing in the map, it makes sense to check the data range in advance:

```
r.info elevation.dem
```

Then we create a reclass rules table `elevation.recl`:

```
1000 thru 1299 = 1 mountainous zone
1300 thru 1499 = 2 subalpine zone
1500 thru 1749 = 3 alpine zone
```

Note that these values are only for illustration and do not reflect the real situation in the Spearfish area.

```
cat elevation.recl
cat elevation.recl | r.reclass elevation.dem\
                   out=elevation.zones title="Elevation zones"
d.rast elevation.zones
```

As another example, we show a mixture of the reclass rules described above. The mixed rules table reclasses the `landuse` map to a reduced number of classes:

```
1 thru 4 = 1 urban area
5 = 2 reservoirs
6 7 = 3 unvegetated and mining areas
8 = 4 transportation and utilities
* = NULL
```

As above, we store this table to a file `landuse.recl` and run `r.reclass` on the `landuse` map:

```
cat landuse.recl
cat landuse.recl | r.reclass landuse out=landuse.simple\
                                      title="Landuses"
```

A hint: To minimize typing efforts, you can start from the category table of the `landuse` map, store it to the file `landuse.recl` and modify it accordingly.

The module `r.cats` outputs the category table; we can redirect it to an initial reclass table:

```
r.cats landuse > landuse.recl
```

Use a text editor to prepare the reclass table from file `landuse.recl`. Be cautious with reclass maps. Since `r.reclass` generates a table referencing the original raster map rather than creating a real raster map, a reclass map will no longer be accessible if the original raster map upon which it was based is later removed. However, `g.remove` prints a warning if such a case occurs. Use `r.mapcalc` to convert a reclass map to a regular raster map:

```
r.mapcalc "landuse.new=landuse.recl"
```

In case that you need to filter areas by size, use `r.reclass.area`:

```
r.report vegcover
r.reclass.area in=vegcover out=vegcover.1000 great=1000
r.report vegcover.1000
```

This script will generate a new map `vegcover.1000` where the minimum area size is 1000 hectares, setting the omitted small areas to NULL. These no-data area could be filled again with surrounding values (`r.neighbors`, e.g. `mode` parameter, see Section 5.4.1).

5.1.6 Assigning category labels

Raster maps may include category labels that are stored in a special category table. Sometimes, this table does not exist, for example, when the map has been created by `r.mapcalc` or when it was imported without labels. In other cases you may want to modify/update existing labels.

Modifying existing category labels for a raster map. The module `r.support` is used to update existing category labels. As an example, we modify existing labels of the `soils` map by replacing some of the soil type acronyms with the full description. First, let us look at the original map:

```
r.report soils
```

Because the map is stored in the read-only PERMANENT MAPSET we cannot modify it (see Section 3.1.2 for explanations). In order to do it, we have to create a copy within our MAPSET:

```
g.copy soils,mysoils
```

The category label editor is included in the module `r.support`. Start it without parameters and enter the map name `mysoils`. Then proceed with <ENTER> to "Edit the category file for [mysoils]?", and enter `y` here. You will

get to the first screen where the highest category number should be defined. Because you are not going to change the number of categories, you can accept the current value and continue with <ESC><ENTER>. This takes you to the category table where you can move around with cursor keys. The table displays the first 10 categories, the rest is on the next page(s). If you want to change just few categories, you can get directly to it by entering it's number into the field "Next category ". As an example, proceed to the category number 24: Enter `24` in that bottom line and hit <ESC><ENTER>. Now you have the requested category number on top of the table which should read "McD". Go to the line and enter the full name "Midway-Razor silty clay loams".[1] You can then go to the category number 10 and then to 20 to learn how it is working. To leave this mode, either scroll through the full table with <ESC><ENTER> or just type `end` into the line "Next category ". Now you get back to the questionnaire mode of `r.support`, you can skip the rest of the questions with <ENTER> to exit the module. Finally, check the updated table with `r.cats` or `r.report`. This procedure looks a bit old-fashioned, but you can use it even remotely through low-bandwidth network access because no graphical user interface is required.

Note that you can create categories for floating point maps representing continuous fields such as elevation or precipitation. To do this, you can run `r.support` and create categories for ranges of FP data. See Section 12.1.3 for an example.

Assigning new attributes to a raster map (⇑). The next application is a bit more sophisticated because we want to automatically assign new attributes to a raster map based on calculations. For illustration, we create a new raster map `fields.areas` by changing the category labels that represent field owners in the map `fields` to labels representing field areas. The procedure will read the map `fields`, calculate the areas for the individual fields, output the results as reclass rules and reclass the fields map according to the rules.

To get the area information, we can use `r.report` or `r.stats`. Compare the results of both (we use `-h` to suppress page headers):

```
g.region -p rast=fields
r.report -h fields units=me
 +----------------------------------------------------------+
 |                    Category Information                  |     square|
 | #|description                                            |     meters|
 |----------------------------------------------------------|
 | 1|C. Smith#1 . . . . . . . . . . . . . . . . . . . . . |    640,000|
 | 2|C. Smith#2 . . . . . . . . . . . . . . . . . . . . . |    600,000|
 | 3|P. Biggam#1. . . . . . . . . . . . . . . . . . . . . |    300,000|
 [...]
```

```
|63|Black Hills Natl. Forest . . . . . |109,420,000|
| *|no data. . . . . |101,220,000|
|-----------------------------------------------------|
|TOTAL                                      |266,000,000|
+-----------------------------------------------------+

r.stats -qan fields
 1 640000.000000
 2 600000.000000
 3 300000.000000
 [...]
```

The `-a` flag allows us to print the area values in square meters related to a category, while `-n` suppresses NULL values. The flag `-q` suppresses printing of percent complete messages to standard output. Note that the area calculation depends on the raster resolution. The results from `r.report` and `r.stats` should be comparable.

However, we cannot use the output of `r.stats` as rules for reclassification of the `fields` map. Since we want to store the area values as category labels for each field we need to modify the output of `r.stats`, using a UNIX tool called `awk`. It is a "pattern scanning and processing language" which is very useful for modification of character strings and simple calculations with data stored in text files (see more details on how to use `awk` in the Appendix A). It allows us to modify a data stream on the fly:

```
r.stats -qan fields
 1 640000.000000
 2 600000.000000
 3 300000.000000
 [...]

r.stats -qan fields | awk '{printf "%d=%d %dm^2\n", $1, $1, $2}'
 1=1 640000m^2
 2=2 600000m^2
 3=3 300000m^2
 [...]

r.stats -qan fields | awk '{printf "%d=%d %d sq meters\n", $1,\
                  $1, $2}' | r.reclass fields out=fields.areas
r.report fields
 [...]
 | 1|C. Smith#1                                          |
 | 2|C. Smith#2                                          |
 | 3|P. Biggam#1                                         |
 [...]

r.report fields.areas
 [...]
```

```
| 1|640000 sq meters                                                    |
| 2|600000 sq meters                                                    |
| 3|300000 sq meters                                                    |
[...]
```

The redirection is done with UNIX piping which sends the output of `r.stats` directly to `awk` to do some formatting, then further on to `r.reclass` for generating the new map. If desired, you can copy the original color table from `fields` to `fields.area` with `r.colors` (see Section 5.1.1).

Clumping raster area features. For some applications we may need to create an individual category number for each raster area (polygon). For example, when the raster map includes several areas with the same soil type (assigned the same category number), we may need to distinguish each area, in case we are interested in computing the size of each area using `r.report`.

The module `r.clump` finds all areas of contiguous cell category values in the input raster map and assigns a unique category value to each such area ("clump") in the resulting output raster map. Assume that we have a simple soils map with 3 soil types in 10 polygons. After "clumping", we will have all 10 polygons numbered individually. Based on that you can assign the individual area sizes as shown above.

```
g.region -p rast=soils
r.report soils units=h
 [...]
 | #|description                                          |  hectares|
 |-----------------------------------------------------------------|
 | 1|Aab. . . . . .                                     |    16.520|
 [...]
 |54|Wb . . . . . .                                     |   375.040|
 | *|no data. . . . . .                                 |   268.680|
 |-----------------------------------------------------------------|
 |TOTAL                                                 |26,600.000|
 |-----------------------------------------------------------------|

r.clump soils out=soils.clump
r.report soils.clump units=h
 [...]
 |  #|description                                        |  hectares|
 |-----------------------------------------------------------------|
 |  1| . . . . . .                                      |     0.160|
 [...]
 |683| . . . . . .                                      |  1405.760|
 |  *|no data. . . . . .                                |   268.680|
 |-----------------------------------------------------------------|
 |TOTAL                                                 |26,600.000|
 +-----------------------------------------------------------------+
```

While some soil names are assigned to several raster polygons in the original `soils` map, all raster polygons in the `soils.clump` map are numbered individually. This is useful, e.g. to calculate area sizes of the individual patches.

5.1.7 Masking and handling of no-data values

Raster MASK allows the user to block out certain areas of a map from analysis by hiding them from sight of other GRASS modules. Effectively, MASK is a raster map which contains the values 1 and NULL. Internally, the maps in use are multiplied pixel-wise with the MASK map. Those cells where the MASK map shows value 1 are available for display and computations while those masked by NULL are hidden. The map name `MASK` is a reserved filename for raster maps, if you have it in your MAPSET, it will be used as a mask in all raster operations.

To create a MASK, you need a base map. This map is used to select which values will represent the hidden and the active areas. As an example, we may decide to work only with areas belonging to "private ownership", stored in the map `owner`. Now start the module

```
r.mask
```

and select menu entry "2 Identify a new mask". It requests a map name with "Enter name of data layer to be used for mask". For our example enter `owner` here. A new screen appears showing the categories of the `owner` map.

```
  IDENTIFY THOSE CATEGORIES TO BE INCLUDED IN THE MASK

   OLD CATEGORY NAME                                        CAT
                                                            NUM

no data . . . . . .                                  0      0____
private ownership . . . . . .                        1      0____
United States Forest Service, Black Hills . . .      2      0____

            Next category: end__  (0 thru 2)
```

You can move around with cursor keys and select "private ownership" by entering `1` on the related line. Continue with <ESC><ENTER> and leave `r.mask` with another <ENTER>. Now, for any map that you display, you will see only the areas belonging to private ownership. Note that at the time of writing this book, the screen shown above listed no-data as 0, which is not correct and will be changed to NULL. Note that this MASK does not apply to vector and sites maps.

You can also generate MASK directly by creating a MASK file with `r.mapcalc` using if-condition as we explain later, or you can rename an existing binary raster map to MASK with `g.rename`.

Finally, you can remove MASK either with `g.remove MASK` or by renaming it to another name for later re-use; for example, `g.rename MASK, mymask`. Also `r.mask` can be used to remove the MASK file.

Zero and NULL value management. GRASS distinguishes between 0 (zero) and no-data (NULL). While zero may represent a true value such as temperature, NULL is used where no value is available. In some situations you may want to modify the current values, such as setting a specific value to NULL or setting NULL to a true value. Here we explain how to exchange NULLs with a single other value. Later on, when talking about `r.mapcalc` we introduce more complex replacement methods.

The NULL values can be managed using the module `r.null`. To change a certain value (e.g. -9999) to NULL, the `setnull` parameter is used:

```
r.null mymap setnull=-9999
```

This will change the value -9999 to NULL in map `mymap`. To replace the NULLs by another number, the `null` parameter is used:

```
r.null mymap null=-9999
```

This will change NULL to -9999. Note that during import of raster data the values to be considered as NULL can be specified. This is important because other GISs may have a different NULL encoding.

Filling data holes in a raster map. Sometimes NULL values appear in map areas where they need to be replaced. This can be done using two approaches:

- replace the NULL values with another value;
- fill the holes according to the hole-boundary values.

The first approach can be done with `r.null` as explained above. For the second option, you can use the script `r.fillnulls`. It will internally fill the holes using interpolated values from the no-data area boundaries using `s.surf.rst` based spline interpolation. That means that the hole boundaries are stored in a separate map which forms a set of "NULL lakes". The values for the "lakes" are interpolated and merged back into the original map. Only the holes are filled with the new values, the original non-NULL values remain unchanged.

It is important to realize that, depending on the shape of the NULL data area(s), problems may occur due to an insufficient number of input cell values for the interpolation process. Most problems will occur if an area containing NULLs reaches the map boundary. You will have to carefully check the result using `r.mapcalc` (generating a difference map to the input map) and/or `d.what.rast` to query individual cell values.

5.2. RASTER MAP ALGEBRA

Raster map algebra is a powerful tool for spatial analysis and modeling using raster data. In GRASS, map algebra is performed with `r.mapcalc`.

In principle, `r.mapcalc` is used in the following way:

```
newmap = expression(map1, map2, ...)
```

where expression is any legal arithmetic expression involving existing raster map layers, integer or floating point constants, and functions known to the calculator. The expression can be provided in a command line mode enclosed within quotes

```
r.mapcalc "newmap=expression(map1, map2, ...)"
```

or you can type `r.mapcalc`, then enter one or more expressions at a prompt (quotes are not necessary), and the expressions are executed after you type end:

```
r.mapcalc
 mapcalc> newmap1=expression1(map1, map2, ...)
 mapcalc> newmap2=expression2(map2, map3, ...)
 mapcalc> end
```

The following **operators** are available in `r.mapcalc`:

%	modulus (remainder upon division)
/	division
*	multiplication
+	addition
-	subtraction
==	equal
!=	not equal
>	greater than
<	less than
>=	greater than or equal
<=	less than or equal
&&	and
||	or
#	color separator into R, G, and B color portions

The following **functions** are available in `r.mapcalc`:

abs(x)	return absolute value of x
atan(x)	inverse tangent of x (result in degree)
cos(x)	cosine of x (x in degree)
double(x)	convert x to double-precision floating point
eval([x,y,...,]z)	evaluates the values of the given expression, pass results to z
exp(x)	exponential function of x
exp(x,y)	x to the power of y

x^y	alternative for x to the power y
float(x)	converts x to single-precision floating point
if	decision operator
if(x)	1 , if x does not equal 0, otherwise 0
if(x,a)	a, if x does not equal 0, otherwise 0
if(x,a,b)	a, if x does not equal 0, otherwise b
if(x,a,b,c)	a, if x > 0, b if x equals 0, c if x < 0
int(x)	converts x to integer [truncates]
isnull(x)	1, if x equals "no data" (NULL)
log(x)	natural log of x
log(x,b)	log of x base b
max(x,y[,z...])	largest of the listed values
median(x,y[,z...])	determines median of the listed values
min(x,y[,z...])	determines smallest of the listed values
mode(x,y[,z...])	determines mode of the listed values
not(x)	1 if x is zero, 0 otherwise
rand(low,high)	generates random number between the values *low* and *high*
round(x)	rounds x to the nearest integer
sin(x)	sine of x (x in degree)
sqrt(x)	square root of x
tan(x)	tangent of x (x in degree)

`r.mapcalc` provides some additional, internal **variables,** which are related to the "moving window" used for calculations:

x()	current x-coordinate of moving window
y()	current y-coordinate of moving window
col()	current col of moving window
row()	current row of moving window
nsres()	current north-south resolution
ewres()	current east-west resolution
null()	NULL value

The value NULL (no-data) is specified with `null()`. As denoted before, NULL differs from 0 (zero).

Integer and floating point data. In map algebra operations, the resulting raster type is defined by the type of the input raster maps and constants. The result of an expression including integer maps and constants will be an integer map; it will be a floating point map if at least one of the constants or input maps is floating point. For example, when dividing two integer maps, it is important to use multiplication by 1.0 to store the result as a floating point map and preserve the decimal values. To illustrate this rule, we will add a constant to an integer map:

```
mapcalc> newmap1 = old_int_map + 123
mapcalc> newmap2 = old_int_map + 123.
```

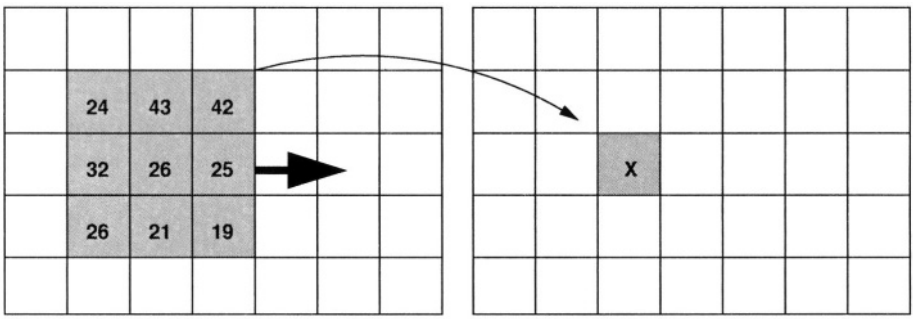


Figure 5.1. "Moving window" method for neighborhood operations in raster map algebra. The raster cell value X of the new map is calculated from a 3x3 matrix of the old map

The resulting map `newmap1` will be stored as integer, while `newmap2` will be stored as floating point. To transform an integer map into a floating point map, simply multiply it by 1.0 or use the float() or double() functions:

```
mapcalc> newmap = 1.0 * old_integer_map
mapcalc> newmap = float(old_integer_map)
```

The calculation of the "normalized difference vegetation index" (NDVI from LANDSAT-TM5) is a good example of an application where the function of integer maps needs to be stored as a floating point map:

```
mapcalc> ndvi = 1.0 *  (tm4 - tm3) / (tm4 + tm3)
mapcalc> ndvi = float( (tm4 - tm3) / (tm4 + tm3) )
```

There is an alternative NDVI algorithm which uses a different function:

```
mapcalc> ndvi2 = 255.0 / 90.0  atan((tm4 - tm3) / (tm4 + tm3))
```

The maps `ndvi` and `ndvi2` are the new floating point raster output maps, `tm3` and `tm4` are LANDSAT-TM5 channels used as integer input maps. Without the multiplication by 1.0, the result would be saved as integer and important information would be lost.

Examples of basic calculations. Cell-wise addition of two or more raster maps is one of the common map algebra tasks. For example, we can add the height of the buildings stored in a map `buildings` to an elevation model stored in a map `dem`:

```
r.mapcalc "dem_with_buildings = buildings + dem"
```

Or we can calculate a weighted average of two maps (here decimal points are used to ensure that the resulting `newmap` is stored in floating point format):

```
r.mapcalc "newmap = (5. * map1 + 3. * map2) / 8."
```

Handling of NULL values in `r.mapcalc`. The basic rule to remember when working with NULL data in map algebra is that operations on NULL cells lead to NULL cells. For example, if one of the maps included in the `r.mapcalc` expression has NULLs in the given area, the resulting map will have NULLs in this area too (both for addition and multiplication functions). In this way, NULL behaves differently from zero, which will have in this area zeroes for multiplication but not necessarily for addition. Therefore, if we want to do operations with NULL data, we need to use a special function `isnull ()`. For example, if we want to fill the NULLs in `map1` with values from `map2` (in other words, when cell value in `map1` is NULL, then write corresponding value of `map2`, otherwise use value in `map1`) we run:

```
r.mapcalc "new=if(isnull(map1), map2, map1)"
```

We can also apply a function of `map2` to the replaced NULL values (in our example we just add a constant 1000.0) as follows:

```
r.mapcalc "new=if(isnull(map1), map2 + 1000., map1)"
```

If you don't use the `isnull ()` function, the NULL values will remain in the output map.

In another example we want to add the maps `map1` and `map2`, where the `map2` contains NULLs. To get a new map with NULLs replaced by 0 (zero) you have to enter:

```
r.mapcalc "new=map1 + if(isnull(map2), 0, map2)"
```

The above examples show that it is important to carefully evaluate the use of the function `isnull()` when applying map algebra to raster maps containing NULLs.

Working with "if" conditions. Various logical operations can be performed with raster data by combining the operators with the `if ()` functions. For example, we can create a new raster `newmap` by applying the `if ()` function to an existing raster `map` and a set of other raster maps or values `a, b, c:`

- *if map = a then b else c* is coded:

```
newmap = if((map == a),b,c)
```

- *if map is not equal a then b else c* is coded:

```
newmap = if((map != a),b,c)
```

- *if map >= a then b else c* is coded:

```
newmap = if((map >= a),b,c)
```

- *if map >= a and map <= b then c else d* is coded:

```
newmap = if(((map >= a) && (map <= b)),c,d)
```

The `if()` functions can be combined to define more complex logical operations:

- Select the values 1 and 2 from `map` and save them in `newmap` while setting all other values to 0:

```
newmap = if((map==1),1,0) + if((map==2),2,0)
```

- Select the values 1 and 2 from `map` and save them in `newmap` as a binary map (only the values 0 and 1, representing "yes" and "no"):

```
newmap = if((map==1),1,0) || if((map==2),1,0)
```

- Conditions with NULL values:
  - Change NULL values into a new value (*if map=NULL then 3 else map*):

    ```
    newmap = if(isnull(map),3,map)
    ```

  - Change all cell values smaller than 5 into NULL value, all other values to 5 *(if map < 5 then NULL else 5):*

    ```
    newmap = if(map<5,null(),5)
    ```

Neighborhood operations with relative coordinates. The usage of the relative coordinates of the `moving window` is another useful option provided by `r.mapcalc` (see Figure 5.1). Neighboring cells can be used in calculations, and larger, possibly asymmetrical moving windows beyond the common 3x3 matrix can be defined. An example:

```
newmap = oldmap[1,1] + oldmap[-1,-1]
```

will read only the offset cells bottom right [1,1) and top left f-1,-1] in an 3x3 matrix for calculating the new map. This option is applicable to different input maps as well. Current row and column values can be integrated into expressions using `row()` and `col()`, the current coordinates of the moving window with `x()` and `y()`.

Note that the offset format is `map [r, c]`, where `r` is the row (y) offset and `c` is the column (x) offset. For example, `map [1,2]` refers to the cell one row below and two columns to the right of the current cell, `map [-2, -1]` refers to the cell two rows above and one column to the left of the current cell, and `map [0,1]` refers to the cell one column to the right of the current cell.

Using `r.mapcalc` on command line. In order to have cursor key support when writing expressions, use `r.mapcalc` on the command line. Remember that the expression must be quoted in this case. To illustrate the usage we will generate a map mosaic using the following rules. If the coordinates in x and y direction are smaller than the given values, then store the values from a map `aspect` in the resulting `newmap`, otherwise store the values from `elevation.dem`:

```
r.mapcalc "newmap = if((x()<599490 && y()<4920855),\
                                  aspect,elevation.dem)"
d.rast newmap
```

In the above example the variable containing the current coordinates of the moving windows was used. Note that color values of the original maps are not transferred.

Creating MASK with `r.mapcalc`. A convenient, non-interactive way to create MASK is to use `r.mapcalc`. As an example, we will create MASK that will allow us to perform operations only in areas with a given range of elevations. We define an expression which assigns the value 1 to the cells that have elevations between 1300 m and 1400 m in the given `elevation.dem` map and NULLs will be assigned elsewhere. In the following sequence of commands, we will display the elevation map (you need to have a monitor open), build the MASK map, display the elevation map again to see the difference, rename MASK to another name to disable it, and display again:

```
d.rast elevation.dem
r.mapcalc "MASK=if(elevation.dem > 1300 &&\
          elevation.dem < 1400, 1, null())"
d.rast elevation.dem
g.rename MASK,maskfile
d.rast elevation.dem
```

You will see that the second display command only shows the selected elevations. All raster map operations performed after setting MASK are performed only in the non-masked areas. After MASK is renamed, the entire elevation map is displayed again.

Evaluation of internal temporary variables. To perform multiple steps within one expression, we can use the function `eval ()`. It evaluates temporary variables without the need to store them in a raster map. The intermediate steps are written within the `eval ()` function, delimited by comma and the last result is saved as a raster map. As a simple example, we will round a floating point map `map1`, store the result for each cell in a temporary variable "a", and subsequently perform a decision based on a condition (if rounded cell value of map==3, then keep 3, otherwise set NULL):

```
r.mapcalc "newmap=eval( a=round(map), if(a==3,3,null()) )"
```

Note that in this simple example the result can be also calculated directly as:

```
r.mapcalc "newmap=if( round(map)==3,3,null() )"
```

In another example we select a range subset from a floating point map, while the NULL values will be kept. This shows a more complex expression using the variables "t1" and "t2" for temporary results (if a<=map<=b then c else d while leaving NULL cells unchanged):

```
newmap  = eval (t1=round(map), t2=if(((t1 >= a) &&  (t1 <= b))\
                                  ,c,d), if(isnull(map), map, t2))
```

Remember that when using the backslash (\) blank must not follow this character.

A useful alternative for value replacement is the module `r.recode` (see Section 5.3.5). For complex value replacements this may be more convenient than writing lengthy "if" statements in `r.mapcalc`.

As a final example of an application with internal variables we can generate a tilted plane, dipping to the northwest with starting altitude 100 m (we have to specify 98 m, as row and column each start with 1):

```
newmap = 98 + row() + col()
```

Many interesting examples of `r.mapcalc` applications for GRASS4.* can be found in Shapiro and Westervelt, 1992.

5.3. RASTER DATA TRANSFORMATION AND INTERPOLATION

GRASS provides capabilities for transformation of raster data to vector and site data model using various approaches, depending on the type of raster data and application. In this section, we also explain how to interpolate raster data.

5.3.1 Automated vectorization of discrete raster data

If the raster data represent linear features or homogeneous areas these features can be transformed to vector data representation. Two modules are available: `r.line` transforms raster line features to vector lines, while `r.poly` transforms raster areas to vector polygons (see Figure 5.2).

Vectorizing lines. It is often necessary to thin (skeletonize) raster lines to a single pixel width using the module `r.thin` before they can be transformed to vector data. The lines are then vectorized using `r.line`. For example, to vectorize the raster map `streams`, we run:

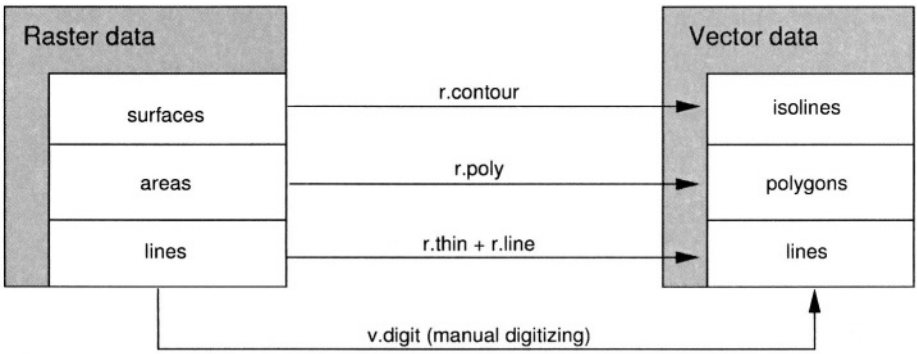


Figure 5.2. Modules for transformation of different types of raster data to vector representation

```
d.rast streams
r.thin streams out=streams.thin
r.line streams.thin out=mystreams
d.vect mystreams col=blue
d.vect streams col=red
```

Note that the `streams` vector map is already available; you may use it for a comparison. Due to the low resolution of the `streams` raster map, the resulting lines have a geometry that follows the original grid pattern, but in spite of this deficiency, the stream network should fit well with the original `streams` vector map displayed in red color. You may try to do the same later with a stream map derived from a higher resolution elevation model.

Vectorizing raster polygons. A different method is needed for vectorization of raster polygons. Such polygons may be generated by reclassification of a raster map (see Section 5.1.5). Using the area borders, we can convert the raster polygons to vector areas. Note that vectorizing of areas does not require thinning. It can be done directly with `r.poly`. As an example, we vectorize the map `elevation.zones` which we created above:

```
d.rast elevation.zones
r.poly -l elevation.zones out=elevation.zones
v.support elevation.zones
d.erase
d.vect.area -r elevation.zones
```

The `r.poly` flag `–l` smoothes corners when generating vector lines. The command `v.support` is required to build the topology of the vector file. We will explain this in more detail in the next chapter. The command `d.vect.area` with flag `–r` can be used to display color-filled vector polygons.

5.3.2 Generating isolines representing continuous fields

If the raster data represent a continuous field, their vector representation is by isolines, or in the case of an elevation surface, contours. The module for generating isolines is `r.contour`. You can let GRASS determine the minimum and the maximum isoline values, and provide only the contour interval by the `step` parameter. To generate contours with a 100 m interval for our `elevation.dem`, we can run:

```
g.region rast=elevation.dem -p
d.rast elevation.dem
r.contour -qn elevation.dem step=100 out=contour100
d.vect contour100
```

The vector topology is built automatically and the resulting contour lines are stored in the vector map `contour100` which can be displayed using `d.vect`.

A related question is how densely the contour lines should be generated. On the one hand they should not hide an underlying map, on the other hand, the information content should be sufficient to get good representation of the surface. In case that there are no additional requirements on the contour interval, the optimal value (`step` parameter) can be computed by the following formula, developed by Imhof (in Hake and Grünreich, 1994:382):

$$A = n * \log n * \tan\alpha \quad \text{with} \tag{5.1}$$

$$n = \sqrt{\frac{M}{100} + 1} \tag{5.2}$$

where A is the contour interval (vertical distance) [meter], α is the slope angle class depending of relief type [in degree], M is map scale [scale number].

The value of α is selected based on the relief type:

- mountainous region: $\alpha = 45°$
- hilly region: $\alpha = 25°$
- plains: $\alpha = 10°$

For example, in a hilly region ($\alpha = 25°$), for a map scale of 1:50,000 the optimal vertical distance is close to 15m. This value can be used in `r.contour` as `step` parameter. To find out the average slope for the selection of α you may calculate the univariate statistics of the `slope` map:

```
r.univar slope
```

The average slope is shown as "Arithmetic mean". We will show later how to generate a slope map from the elevation data. The mathematical descriptions of the equations behind `r.univar` are given in the Appendix B.

5.3.3 Raster data transformation to sites

For some applications, it is useful to transform the raster map into a sites (points) map. GRASS provides the module `r.to.sites` for this purpose. Use the –a flag to output the cell values as a decimal attribute rather than a category number (which would be an integer).

The density of sites is controlled by the current region GRID RESOLUTION which you can adjust with `g.region`. To see how it works, we generate a sites map from the raster map `transport.misc` as follows:

```
g.region res=30 -p
d.erase
d.rast transport.misc
r.to.sites -a transport.misc out=transport.misc
d.sites transport.misc
```

The sites are shown per default as grey crosses which can be adjusted as described in Chapter 7 about site data processing. You can learn about additional methods for generating sites from raster data, such as `r.random`, in this chapter as well.

5.3.4 Interpolation of raster data and resampling

Transformation of a raster map layer to a different resolution is done internally whenever the region settings are changed. It is done by simple resampling which assigns the same values found in the original map to the cells in the resampled map, leading to a discontinuous surface. This approach has been designed for rasters representing classes. When changing resolution of rasters that represent continuous fields, interpolation should be used (Figure 5.3). Interpolation is also needed when filling gaps in merged raster data or when raster data are patchy and contain NULL values that need to be replaced to achieve continuous coverage.

Inverse distance weighted average (IDW) interpolation. Two modules for IDW interpolation of raster data are available: `r.surf.idw` and `r.surf.idw2`. Both modules compute the value at a grid point as a weighted average of a given number of neighboring grid points (default number of points is 12). The weight is depends on a distance between the computed grid point and the given point (see the equation in Appendix B.2). The main difference between the two modules is in the search method used for finding the neighboring grid points. The module `r.surf.idw` uses a more efficient approach and it also supports interpolation of data in geographic coordinates (latitude-longitude). A new resolution for the resulting raster map should be set using `g.region` before running the interpolation command.

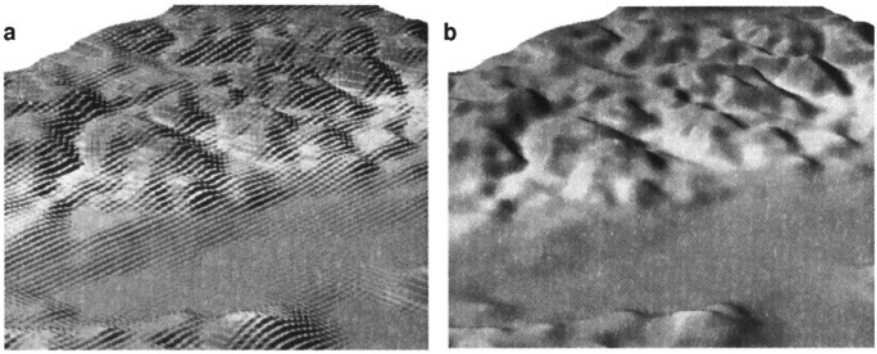


Figure 5.3. Difference between a) resampling and b) interpolation to higher resolution by RST

Regularized spline with tension (RST) interpolation. If there are large gaps in the raster data, it is advisable to use splines for the interpolation. If appropriate parameters are selected, the result will be much better than with the IDW method. The splines interpolation is available for raster, vector and site data, as `r.resamp. rst`, `v.surf.rst` and `s.surf.rst`. Because in GRASS 5.3 (which has been used in this book) the raster version `r.resamp.rst` was not sufficiently tested, we recommend transforming the raster map into sites, using `r.to.sites` or `r.random` (see Chapter 7.1.2 and Chapter 12) and using the `s.surf.rst` module. As in the previous case, the new resolution is selected using `g.region` before the module is executed.

Besides the interpolation, the `s.surf.rst` module can also calculate topographic parameters such as slope angles, aspect angles and curvatures (profile, tangential and mean). For details about the spline interpolation see the Section 7.3, Appendix B.2 and papers by Mitas and Mitasova, 1999, Mitasova and Mitas, 1993 and Mitasova and Hofierka, 1993.

When using interpolation it is important to check the quality of the resulting map, for example by computing the difference between the input and output raster maps and by viewing the associated slope and aspect maps to identify possible artifacts.

Bilinear interpolation. The bilinear interpolation is performed with `r.bilinear`. The interpolation function with constant gradient (a plane) is derived from the values of 4 points defining the cell centers in a given rectangular area. It can therefore be applied only to the raster maps with no NULL data values. The gradient of the interpolation function changes discontinuously across lines intersecting the cell centers of the input raster, so the method is useful only for small changes in resolution. For larger changes

in resolution, a check board structure may appear. On the other hand, the method is very fast and simple and has a special option for "wrap-around" interpolation of latitude-longitude rasters.

Resampling. GRASS uses automatic resampling when the region resolution is different from the resolution of the given raster map. Resampling can be also applied to a raster map by `r.resample`. When resampling from lower to higher resolution, the high resolution cells are assigned the same values as the cell within which they are located. When resampling from higher to lower resolution, the low resolution cell is assigned the value of the high resolution cell which is located the closest to its center (nearest neighbor). Resampling is designed for raster data which represent geometrical features, such as lines and areas (raster maps with categories). If applied to raster maps representing continuous fields, the resulting surface may have a checkerboard pattern (see Figure 5.3).

5.3.5 Recoding of raster map types and value replacements

Sometimes we need to convert between different raster map types. The module `r.recode` has routines for conversion between every possible combination of raster type (e.g. INT to DOUBLE, DOUBLE to FLOAT, etc). The recoding is based on the rules which are read from standard input (i.e., from the keyboard, redirected from a file, or piped through another program). Standard output floating point raster data precision is FLOAT, with -d DOUBLE precision will be written. The general form of a recoding rule is:

```
oldmin:oldmax:newmin:newmax
```

To simply convert a raster between types, for example from INT to FLOAT, run the command with the following rule:

```
r.recode in=intmap out=fpmap << EOF
200:1500:200.0:1500.0
EOF
```

This will convert an INT raster map to a new FLOAT raster map with the same range of values. To convert the map from INT to DOUBLE while simultaneously changing the range of values we can use:

```
r.recode -d in=intmap out=doublemap << EOF
200:1500:0.0005:0.000008
EOF
```

You will find additional alternatives for defining the rules in the `r.recode` manual page. Note that in the above examples we use another UNIX method to direct data such as the recode rules into the command `r.recode`. This

method is very convenient, especially for script programming. In the first line EOF (end of file) is specified, the module reads input unless the second EOF appears.

Value replacement. In order to replace existing cell values with different ones, you can again use the module r.recode. The formatting of the recoding rules is the same as described above. In the following example, the old values 1, 2 and 3 are replaced by 1.1, 7.5 and 0.4 respectively:

```
r.recode in=oldmap out=newmap << EOF
1:1:1.1:1.1
2:2:7.5:7.5
3:3:0.4:0.4
EOF
```

Each value appears twice because the module expects ranges for the old and new values to be specified. This value replacement method is often faster than formulating complex "if" conditions with raster map algebra through r.mapcalc.

5.4. SPATIAL ANALYSIS WITH RASTER DATA

A wide range of spatial analysis tasks can be performed using GRASS raster modules. Map overlay, generation of buffers, finding of shortest paths, and deriving topographic parameters can be combined to analyze complex spatial relationships. We describe some of the modules in the following section.

5.4.1 Map statistics and neighborhood analysis

To compute univariate statistics for a raster map use r.univar. It computes the number of cells, minimum, maximum, range, arithmetic mean, variance, standard deviation, and the variation coefficient. As an example, we apply it to the map elevation.dem:

```
g.region -p rast=elevation.dem
r.univar elevation.dem
 [...]
 Number of cells: 292317
 Minimum: 1066
 Maximum: 1840
 Range:  774
 Arithmetic mean:  1353.67
 Variance:  31343.4
 Standard deviation: 177.041
 Variation coefficient: 13.0786 %
```

Please refer to Appendix B for equations. The area statistics for each category or floating point range is computed by `r.stats`, which we have already described in Section 5.1.6.

Neighborhood analysis. The neighborhood operators determine a new value for each cell as a function of the values in its neighboring cells. All cells in a raster map layer, except the cells at the map boundaries, become the center cell of a neighborhood as the neighborhood window moves from cell to cell throughout the map layer. The following neighborhood operators, with user defined sizes of the moving window, are available in `r.neighbors` (see Appendix B for equations):

- average: the average value within the neighborhood;
- median: the value found half-way through a list of the neighborhood's cell values, when these are arranged in numerical order;
- mode: the most frequently occurring cell value in the neighborhood;
- minimum: the minimum cell value within the neighborhood;
- maximum: the maximum cell value within the neighborhood;
- stddev: the statistical standard deviation of cell values within the neighborhood (rounded to the nearest integer);
- sum: sum of cell values within the neighborhood;
- variance: the statistical variance of cell values within the neighborhood (rounded to the nearest integer);
- diversity: the number of different cell values within the neighborhood;
- interspersion: the percentage of cells containing categories which differ from the category assigned the center cell in the neighborhood, plus 1.

As an example, we can compute a simple "biodiversity" map as follows:

```
r.neighbors vegcover out=veg.diversity method=diversity size=5
d.rast veg.diversity
d.legend -m veg.diversity
```

You can experiment with different neighborhood window size to see its impact on the resulting map. Please refer to the manual page of `r.neighbors` for further details.

To find an average of values in a cover map within areas assigned the same category numbers in a base map, you can use `r.average`. For example, we can compute the average elevation for each field as:

```
r.average base=fields cover=elevation.dem out=elevation.byfield
d.rast elevation.byfield
```

To compute the average of values which are stored as category labels, we need to use the flag –c. In this case, the category label for each category in the cover map must be a valid number (integer or decimal). For example, to compute the average K factor (soil erodibility factor) for each field, we run:

```
r.average -c base=fields cover=soils.Kfactor out=Kfact.byfield
d.rast Kfact.byfield
```

A module working in a similar manner is `r.median` which finds the median of values in a cover map within areas assigned the same category value in a user specified base map.

The area of a surface represented by a raster map can be computed by `r.surf.area` which calculates both the area of the horizontal plane for the given region and an area of the 3D surface estimated as a sum of triangle areas created by splitting each rectangular cell by a diagonal.

Volume calculation. To compute volume of an object defined by a surface (or its subsection defined by clumps) and a horizontal plane, you can use `r.volume`. In our example, we assume that a football stadium with a parking lot and related facilities will be built in Spearfish. For the construction the earth's surface has to be excavated to a minimum depth of 5 m. To calculate the costs for the excavation work, we have to find the corresponding soil volume. The corners of the stadium area may be known from GPS measurements. To create a raster map of this area, store the coordinates of the stadium in the file `stadium.txt` in the following format (UTM coordinates for an area within Spearfish city):

```
A
591316.80   4926455.50
591410.25   4926482.40
591434.60   4926393.60
591341.20   4926368.70
= 1 stadium
```

This will define a raster area by its corner points labeled as "1 stadium". As we operate only in Spearfish city, you may zoom into the city area before continuing. To zoom and import the raster polygon coordinates file, run:

```
d.rast roads
d.zoom
g.region -pa res=1
r.in.poly in=stadium.txt out=stadium
d.erase
```

```
d.rast roads
d.rast -o stadium
```

The flag –a aligns the region to the resolution, i.e. slightly extends it so that the resolution is a whole number. The resulting map contains the desired stadium area. We can now calculate the volume of the material that needs to be taken out (note that the map `elevation.dem` is only at 30 m resolution and an integer map, so the results will not be very accurate). Because we need to compute the volume only for a certain subarea, we will first set MASK to the area defined by the raster map stadium. Then, we "normalize" the excavation to have the bottom of the volume at zero elevation. This can be done easily by subtracting the desired minimum elevation from the DEM. We can find the minimum by running e.g. `r.univar` after masking all areas except the stadium:

```
g.copy stadium,MASK
r.univar elevation.dem
 [...]
 Minimum: 1120
 Maximum: 1139
 [...]
```

When using MASK the elevation statistics is calculated only for the stadium area. We store the portion of the DEM as defined by the current region into a new map and subtract the minimum height of the terrain and the desired excavation depth. Because part of the volume is below zero elevation now, we "lift" it up to achieve zero elevation as minimum. Then we can calculate the volume:

```
r.mapcalc "excavation=elevation.dem - 1120 + 5"
d.rast excavation
r.univar excavation
 [...]
 Minimum: 5
 Maximum: 24
 [...]

r.volume excavation
Cat     Average    Data    # Cells       Centroid           Total
Number in clump   Total   in clump Easting    Northing    Volume
   1      12.17   107461   8832     591375.50 4926425.50 107461.00
                                          Total Volume = 107461.00

g.remove rast=MASK
```

The soil volume to be excavated for the stadium is roughly 107,400 m^3.

Cross-category reports. We can easily generate reports for two or more maps which include occurrence of categories in the second map for each category in the first map. As an example, we can create a report which includes the areas in each land use category listed for each field owner category:

```
g.region -p rast=fields
r.report fields,vegcover units=h
 [...]
 |MAPS: SCS farmfields (fields in PERMANENT)                    |
 |        Vegetation Cover (vegcover in PERMANENT)              |
 |--------------------------------------------------------------|
 |                      Category Information        |           |
 | #|description                                    |  hectares|
 |--------------------------------------------------------------|
 | 1|C. Smith#1                                     |    64.000|
 |  |----------------------------------------------|----------|
 |  |1|irrigated agriculture. . . . . .  |     4.000|
 |  |2|rangeland. . . . . .  |    57.000|
 |  |6|disturbed. . . . . .  |     3.000|
 |--------------------------------------------------------------|
 | 2|C. Smith#2                                     |    60.000|
 |  |----------------------------------------------|----------|
 |  |1|irrigated agriculture. . . . . .  |    48.000|
 |  |2|rangeland. . . . . .  |     8.000|
 |  |4|deciduous forest . . . . . .  |     4.000|
 [...]
```

If you want to build a table using a set of maps, you can select them with `g.mlist` and send the list to the module in one line:

```
g.mlist type=rast pattern="soil*" sep=,
MAP=`g.mlist type=rast pattern="soil*" sep=,`
r.report map=$MAP
```

The ` ` characters have a special meaning in a shell (on the command line or in scripts). A command enclosed by these characters is executed and the message sent by the command can be stored in an environmental variable (in our case MAP) as above.

5.4.2 Overlaying and merging raster maps

As we have already mentioned, it is possible to overlay raster maps visually in the GRASS monitor using `d.rast -o`. However, to store such a raster map overlay in a new map, a different method is needed. To merge two maps into a single new map, use `r.patch`. It requires the input maps and a name for the new output map. The specified input map order determines the result: The NULL areas in the first map (which is on top of the virtual map stack) are filled with values from the second map and so forth for further maps. Overlaying or

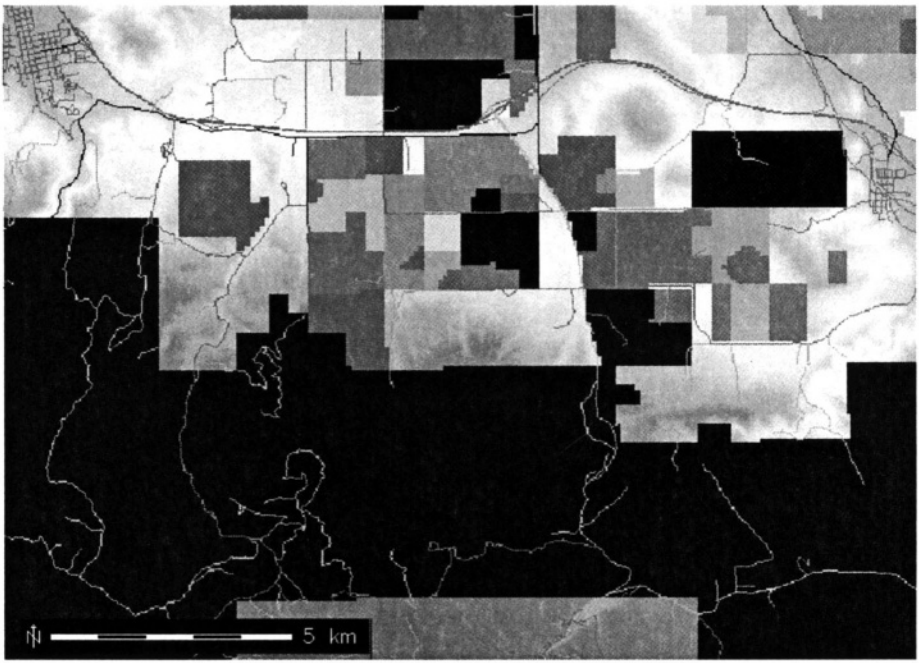


Figure 5.4. Map composite of roads, land use map and elevation model created with `r.patch` (Spearfish area)

combining adjacent image raster maps with large number of colors is a slight problem, because the GRASS display color model significantly slows down when more than 8000 colors are used in a map. In case of 24 bit images, such as color aerial photos, the color channels R, G, B are usually kept separated. A workaround for patching high-color maps together without slowing down GRASS is the script `i.image.mosaic`. It merges the color tables of adjacent maps accordingly while patching the maps.

For example, multiple adjacent digital elevation models can be imported and merged as follows:

```
r.in.arc in=392619.asc out=392619.dem12
r.in.arc in=392620.asc out=392620.dem12
r.in.arc in=392626.asc out=392626.dem12
r.patch in=392619.dem12,392620.dem12,392626.dem12 out=dem12.5
```

To modify the color table to a suitable one for a DEM (from green over yellow to red), run:

```
r.colors map=dem12.5 col=gyr
d.rast dem12.5
```

In another example, we will compose several maps within the same area. Except for the Spearfish map `elevation.dem`, all maps contain NULL values which are filled by the underlying map(s) (in case that they contain non-NULL values in these particular cells):

```
r.patch in=roads,railroads,fields,elevation.dem out=map.comp
d.rast map.comp
```

Here, the roads network is on top, followed by the other maps. The DEM is filling all areas not being filled by other maps (see Figure 5.4). Map overlays can also be done with `r.mapcalc`. The "r.mapcalc tutorial" (Shapiro and Westervelt, 1992) describes several examples.

Figure 5.5 shows the differences between a map merge using `r.patch` or `r.mapcalc` respectively. The module `r.patch` patches on a basis of overlays, while `r.mapcalc` combines the raster map layers base on a user defined expression, as described in the Section 5.2.

For validation, the module `r.cross` checks the plausibility of merged maps against the input maps. Using the interactive mode of `r.cross`, you first have to enter the names of the source maps which have been merged earlier (at least two). When all source maps are specified, press <ENTER> one more

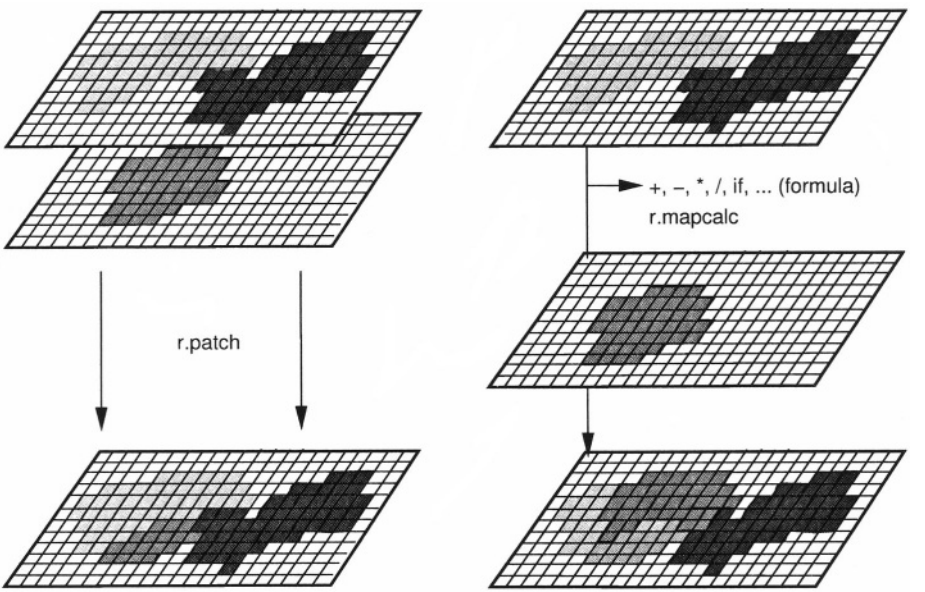


Figure 5.5. Raster data merging with `r.patch` (left) and `r.mapcalc` (right)

time to finish the dialog mode. Now you have to specify a name for the output control map containing the resulting validation. This map can be displayed and checked using `d.what.rast` to see whether the results are equivalent to the source data or not. In case that you want to display only the table with category labels (attributes), use `r.cats` in conjunction with the `r.cross` output map.

5.4.3 Buffering of raster features

A method to widen linear or area features in space is buffering. The implementation in the GRASS `r.buffer` module allows us to define one or several buffers with different spatial extents.

As an example, we want to find out the noise distribution along the interstate for the Spearfish area. To illustrate the use of buffers we use a simplified model which does not take wind into account. We want to know which residential areas are influenced by different levels of noise. The result may determine whether noise protection walls have to be installed or not. The buffer zones may be 250 m (high disturbance), 500 m (moderate disturbance) and more than 500 m (low disturbance).

First, we have to select the interstate and store it into a new map. This can be done, for example, with `r.mapcalc` (if-condition) or `r.reclass`. Here we use the latter command to demonstrate another way of entering the class values. First, we check the category value of the interstate with `r.report`, and then we create the raster map `interstate`:

```
r.report roads
r.reclass roads out=interstate << EOF
 1 = 1 interstate
 EOF
d.rast interstate
```

We can now apply the buffer zones:

```
r.buffer interstate out=interstate.buf dist=250,500,2500
d.rast interstate.buf
r.poly interstate.buf out=interstate.buf
v.support interstate.buf
```

We have selected the last buffer to be 2,500 meters in order to reach "quasi infinity" (of course, that's highly dependent of the wind direction). Finally, we have to intersect the buffered interstate map with the `landuse` map which contains the residential areas:

```
r.report landuse
r.mapcalc "noise=if(landuse==1, interstate.buf, null())"
```

Using some additional commands, we can look at the noise map. The numbers as drawn by `d.rast.labels` represent the categories 2 (0 m to 250 m),

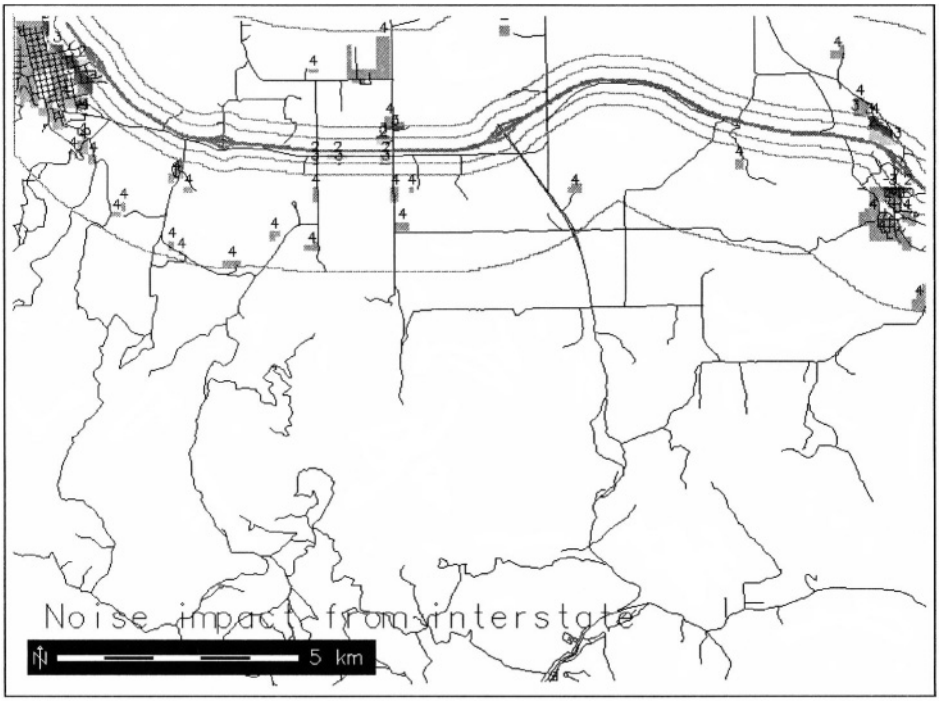


Figure 5.6. Spearfish noise impact map from interstate (simple noise buffer model)

3 (250 m to 500 m) and 4 (500 m to 2500 m) which we have used for the buffering:

```
d.rast noise
d.vect roads col=grey
d.rast -o interstate
d.vect interstate.buf col=green
d.rast.labels noise col=black attr=cat
echo "Noise impact from interstate" | d.text col=black line=18
d.barscale at=5,90
```

The resulting map shows only those residential areas which are influenced by the interstate according to the noise definition (see Figure 5.6). You can also query the map directly or print a report of affected residential area sizes (filtering all NULL values):

```
d.what.rast
r.report -n noise units=h
```

5.4.4 Cost surfaces

Cost surfaces are raster maps showing the cumulative costs of moving between different geographic locations in an input raster map. Since GRASS 5.3 does not provide vector network analysis tools (but GRASS 5.7 will), this is an approach to reach similar functionality. Each raster cell in the input raster map will contain a value which represents the cost of traversing that cell. `r.cost` will produce an output raster map layer in which each cell contains the lowest total cost of traversing the space between each cell and user-specified points.

Let us start with an application – assume a wildfire in the Spearfish region at coordinates 597192E and 4915574N reported by an automated system. The fire was detected near an unimproved road in the Black Hills National Forest. Our task is to notify the fire brigade either in Spearfish (city north-west in our area) or Whitewood (village north-east in our area). The decision is influenced by two parameters: the potential speed to reach the destination depends on the roads condition and the distance to the fire. To solve the problem we have to calculate a "cost surface".

The roads map is available as a vector map. We transform the vector map to a raster map with 50 m resolution. For that, we change the current resolution to 50 m, then transform the vector map to a new raster map:

```
g.region vect=roads res=50 -pa
d.erase
d.vect roads
v.to.rast roads out=roads50m
d.rast roads50m
```

Note that the optional flag –a of `g.region` aligns the region boundary coordinates according to the resolution: the current region is slightly expanded to get a whole resolution number, in this case 50 m.

To store the location of the fire in a fire map, we use a sites module `s.in.ascii` which will be explained later in more detail in Chapter 7:

```
echo "597192 4915574 1" | s.in.ascii sites=fire
d.sites fire col=red
```

The `echo` command is a UNIX command to display a line of text. In this case we pipe the coordinates string to the module `s.in.ascii` which is effectively the same as storing the coordinates string into a text file and subsequently importing it. Next we have to reclass the `roads50m` map to obtain a map of potential specific travel time. This may be the inverse of the potential speed in km/h depending on the road quality. The reason to use the specific travel time instead of the speed is that `r.cost` considers high values as costly.

For the potential speed, you can use `r.report` to get the roads quality labels. Since reclassifications would only lead to integer maps, we use

`r.recode` to replace the road quality in `roads50m` with the specific travel time (which is a floating point value) in a new map. An example to understand the specific travel time: Considering 100 km/h as potential speed on the interstate for a fire brigade (label 1 in `roads` map), we calculate 1/100 = 0.01 h/km as specific travel time. To store these recode rules, we create a table `roads.recode` for all calculated values. This will replace the roads quality values with the specific travel time in a new map (refer to Section 5.3.5 for details):

```
1:1:0.0100:0.0100
2:2:0.0125:0.0125
3:3:0.0167:0.0167
4:4:0.0200:0.0200
5:5:0.0500:0.0500
```

Based on these recode rules, we generate a new map `roads.travel`:

```
cat roads.recode | r.recode roads50m out=roads.travel
r.report roads.travel
```

To make the map more meaningful, we define a new color table with high potential speed represented as green and the lowest potential speed as red with yellow in between:

```
r.colors roads.travel col=rules
 0.0 green
 0.0167 yellow
 0.1 red
 end

d.rast roads.travel
d.sites fire col=red
```

To ensure that the full data range is covered, we define a lower specific travel time than the actual minimum value as green and a higher value as red. The value in the middle is taken from the above recode table.

The new map of specific travel time is the input to the cost surface module, which also requires the location of the wildfire. The costs to travel along the roads based on the inverse speed are then calculated from this location. Again, we redefine the color table for cost map, to make the map more communicative:

```
r.cost -kv in=roads.travel out=roads.cost coord=597192,4915574
r.colors roads.cost col=gyr
d.rast roads.cost
d.sites fire col=red
```

The fire brigade in Spearfish may be located at 590772.6E and 4926787.3N while for Whitewood it may be at 608297.8E and 4924206.0N. By visual inspection of the resulting cost map, mouse querying or using `r.what` you can

find out which fire brigade is closer in terms of distance and potential speed to reach the fire. The result depends on the parameters used to prepare the input maps. Using `r.what` is probably the best idea, as we can directly enter the above coordinates of the fire brigades:

```
r.what roads.cost east_north=590772.6,4926787.3
  590772.6|4926787.3||7.8173332214
r.what roads.cost east_north=608297.8,4924206.0
  608297.8|4924206.0||7.7000617981
```

The result tells us that the fire brigade in Whitewood can reach the wildfire slightly faster (costs are lower) according to our definitions.

Of course we are also interested in getting the optimal route through the route network. The optimal routing module `r.drain`, which analyzes the cost surface, needs the coordinates of the fire brigades. Additionally we specify the flag `-n` to count the number of cells along the path (a simple indicator for the distance):

```
r.drain -n in=roads.cost out=spear.route\
                    coord=590772.6,4926787.3
d.rast spear.route
r.drain -n in=roads.cost out=white.route\
                    coord=608297.8,4924206.0
d.rast white.route
d.vect roads col=red
```

This delivers two path maps with accumulated numbers of cells, indicating the distance. Precise distance measure could be done by converting the raster lines to vector lines and generating a related line length report.

Note that `r.cost` only calculates along non-NULL values. In case that a starting point falls into a NULL area in the input map, you have to replace the NULL values with something meaningful for that application (in the above example a minimum speed). Either use `r.mapcalc` with an `isnull ()` function to replace the NULL values, especially when you want to replace the NULLs with another underlying map, or use the `r.null` module when replacing the NULLs with a single value.

In our example we did not take into account that Deadwood, the closest town to our wildfire, is located south-east of the area. Another drawback is that some roads that appear connected on the raster map do not cross in reality, for example, if there is an overpass over the highway.

Computing a distance map. To compute the shortest distance of each pixel from raster lines, such as determining the shortest distances of households to the nearby road, you can use cost surfaces with cost value 1. The calculation is done with `r.cost` as follows (example for Spearfish region):

```
g.region rast=roads -p
r.mapcalc "area.one=1"
r.cost -k input=area.one output=distance start_rast=roads
r.mapcalc "dist_meters=distance * (ewres() + nsres())/2."
d.rast dist_meters
d.zoom
d.rast.num dist_meters
```

The map `area.one` is used as cost information, for calculating distances we use costs of 1. The distances are calculated from the roads network. Note that `d.rast.num` which prints the cell values as text labels into raster cells, only works when zooming into a map portion. The resulting `distance` map carries distance information in number of cells to the closest road. To calculate a real distance map (in meters), the cell values have to be multiplied with the cell resolution. In our example, we assume square pixels with identical resolution in north-south and west-east directions.

5.4.5 DEM and watershed analysis

Topographic analysis (or surface analysis) provides a wide range of parameters that describe the geometrical properties of the studied surface, such as:

a) summary parameters and profiles, for example volumes, surface areas, roughness, fractal dimension;

b) point parameters describing the geometry of the surface in the given point such as slope, aspect and different types of curvatures;

c) flow parameters based on integration along flowlines, such as slope length, upslope contributing area, watershed (basin), stream network;

d) ray tracing parameters based on lines (rays) emitted from or towards the surface, such as line of sight, insolation.

We will cover a subset of the above mentioned tasks in the following paragraphs.

Slope, aspect and curvatures. The module `r.slope.aspect` generates raster map layers of slope, aspect, curvatures and partial derivatives from a raster map layer of true elevation values. The slope values are calculated per default in degrees, or you may change the units to percentage using the parameter `format`. In case you are working in a coordinate system that uses feet (or units other meter), the module automatically converts the horizontal distances to meters. You must use parameter `zfactor` to convert elevation to meters to obtain correct values of slope, curvatures and derivatives. The aspect values represent the direction of flow (they point downslope) measured in degrees from east, increasing counterclockwise: 90° is north, 180° is west, 270°

is south and 360° is east. Additionally, you can calculate a profile curvature map (measured in the direction of the steepest slope) and a tangential curvature map (measured in the direction of the contour tangent). For exact definition of slope, aspect and curvatures see the equations in the Appendix B.3. Further explanation and applications of these parameters in relation to DEM quality assessment and modeling of processes is in Chapter 12. For general explanation of curvatures see, for example, Alexandrov et al., 1989 or Mitasova and Hofierka, 1993.

The module computes topographic parameters based on approximation of the terrain surface by a second order polynomial. This leads to the computation of partial derivatives, needed for estimation of slope and aspect, as weighted averages of elevation differences in the 3x3 neighborhood of the given grid point (see Appendix B.3). The equations are also known as Horn's formula (Horn, 1981). When applied to DEM represented by integer meters the aspect is biased in certain directions and reinterpolation to floating point DEM is recommended (see Section 12.1).

For example, to compute slope, aspect and profile curvature using the elevation map in the Spearfish data set simply run:

```
r.slope.aspect el=elevation.dem slope=slope asp=aspect\
                                              pcurv=profcurv
```

Find more application examples of this command in Chapter 12.

Flow parameters and watersheds. Flow parameters are derived by flow-tracing and are "integral" rather than "point" parameters (the value at each cell is computed based on a sum of values from other cells – in our case along the flowpath). Flow routing is based on flowlines, which are curves perpendicular to contours with direction given by aspect (minus gradient). The basic flow parameters are:

- watershed (basin) areas (upslope area for a given outlet);
- stream network;
- flowpath length (also hillslope length);
- flow accumulation (also flowline density, used to compute upslope contributing area).

Numerous algorithms have been developed for flow routing, based on the approach for estimation of the steepest slope direction and distribution of "water" to the downslope cells:

- single flow direction (SFD, D8) moves flow into a single downslope cell, it is used by `r.watershed` and `r.terraflow`;

- D-infinity or vector-grid approach used by `r.flow`;
- multiple flow direction (MFD) partitions the flow into 2 or more downslope directions, it is used by `r.terraflow`; `r.topmodel`, and the `r.mapcalc` flow implementation described in "r.mapcalc tutorial" (Shapiro and Westervelt, 1992);
- bivariate (2D) flow used by `r.hydro.CASC2D` and an experimental module `r.sim.water`.

See Chapter 12 for examples of the practical applications of flow routing and comparison of the different approaches.

Watershed basin analysis is performed with `r.watershed`. As an example we calculate watershed basins from the elevation map in Spearfish region:

```
r.watershed el=elevation.dem basin=basins thresh=1000\
                                          stream=rivers
d.rast rivers
d.rast basins
d.rast.labels basins
```

The threshold parameter is required to define the minimum basin size through cell numbers or overland flow units. The stream segment values correspond to the watershed basin numbers. Basins which are incomplete within the current region are omitted. This module does not require filling of depressions in DEM prior to its application because it uses shortest path algorithm to traverse the elevation surface to the outlet. In applications to new type of DEMs, based on LIDAR or radar surveys, this often leads to more accurate results compared to traditional methods.

For the flow accumulation and drainage direction analysis, the `r.watershed` module provides the option to use a binary depression (sinks) input map which contains lakes or other depressions in landscape that are large enough to store surface runoff. Such a depression map can be generated with `r.fill.dir` within two steps. First the DEM is filled, then the resulting filled DEM is subtracted from the original elevation map using `r.mapcalc`, with the result binarized on the fly using if-condition. The binary difference map is the depression map for `r.watershed`. In some cases, such as for the Spearfish elevation map, it is necessary to run `r.fill.dir` repeatedly (using output from one run as input to the next run) before all depressions are filled:

```
r.fill.dir in=elevation.dem elev=elevation.fill1 dir=direct1
r.fill.dir in=elevation.fill1 elev=elevation.fill2 dir=direct2
r.fill.dir in=elevation.fill2 elev=elevation.fill3 dir=direct3
r.mapcalc "depressions.bin=if((elevation.dem - \
                    elevation.fill3)< 0., 1, 0)"
d.rast depressions.bin
```

Because the D8 algorithm used in `r.watershed` is not suitable for dispersal flow on hillslopes, the module `r.flow` should be used for applications where overland flow pattern is needed. The program allows us to trace flow upslope or downslope and compute raster map layers representing flowline accumulation, flowline length, and a vector map for flowlines.

For computation of flow accumulation from massive DEMs that cannot be handled by `r.watershed` use the new `r.terraflow` module (Arge et al., 2003). This module is also useful for generating smooth flow pattern over hillslopes using the MFD option with the possibility to switch to D8 single flow direction (SFD) routing for extracting the streams. You can learn more about flow tracing and watershed analysis in Chapter 12, which includes several applications related to land use management.

Geomorphology. The GRASS module `r.param.scale` extracts terrain parameters from a DEM with parameter `feature`. The morphometric features which are calculated are peaks, ridges, passes, channels, pits and planes. The resulting map depends on the moving window size. As an example, we extract morphometric features from the Spearfish elevation map:

```
r.param.scale elevation.dem out=morphology size=7 param=feature
d.rast morphology
d.legend -m morphology
```

The resulting map contains the above mentioned categories. If you plan to use this module for a serious research or applications we suggest that you read Wood, 1996, which provides detailed explanations and equations used in the module.

Sun illumination and solar energy maps. Many earth processes are influenced by the received solar energy. In environmental modeling, incoming radiation is needed as an input for evapotranspiration models; in urban planning and design, it is important for designing buildings and parks. For solar illumination effects and potential radiation calculations, GRASS provides two modules: `r.sunmask` to calculate a cast shadow map, and `r.sun` to calculate solar radiation (irradiance and energy) maps.

The module `r.sunmask` uses the SOLPOS2 algorithm from NREL (National Renewable Energy Laboratory) to calculate the position of sun in the sky for a given date, time, and a location on earth. You may use this feature for other purposes in case you need the sun position (`r.sunmask` provides a flag `-s` to calculate the sun position and exit then). For example, we can compute the cast shadow map in our Spearfish area for November 29, 2001 at 11:15h as:

```
r.sunmask -v elevation.dem out=shadow year=2001 month=11\
            day=29 hour=11 minute=15 sec=0 timezone=-7
```

As mentioned, the sun position and date calculation results from `r.sunmask` may be used as input for `r.sun` (Hofierka and Súri, 2002b, Súri and Hofierka, 2004), which requires the latitude and the day-of-the-year for the given location as input:

```
r.sun elevin=elevation.dem aspin=aspect slopein=slope lin=2.5\
 alb=0.2 beam_rad=b80 diff_rad=d80 refl_rad=r80 insol_time=it80\
 lat=44.4342 day=80
```

The radiation output maps `b80`, `d80`, and `r80` contain direct (cloudless direct beam radiation), diffuse, and reflected radiation for the given day, respectively. The sunshine duration is recorded in `it80` map. Optionally you can incorporate a shadowing effect of terrain using the `-s` flag. In mountainous and even hilly areas this can lead to very different results! Note that calculating the shadowing effect of relief can be computationally demanding.

Recall that you can get the map center coordinates for the current region with:

```
g.region -c
g.region -l
```

The first flag `-c` reports these coordinates in the current coordinate system, while the second flag `-l` reports in latitude-longitude. See Section 12.2.3 later on for another application with `r.sun`.

Synthetic DEMs. To support terrain analysis, GRASS provides the module `r.surf.fractal` to create synthetic elevation models of selected characteristics (see Wood, 1996). The elevation models are created based on a given fractal dimension (Mandelbrot, 1983). This dimension D lies between the Euclidian dimensions 2 (plane) and 3 (volume), the closer D is to 3 the more rugged is the generated relief. We can compute a synthetic DEM with dimension 2.01 and display it as a shaded relief map as follows:

```
g.region res=50 -p
r.surf.fractal out=fractdem.201 d=2.01
r.info fractdem.201
r.colors fractdem.201 col=gyr
d.rast fractdem.201
r.slope.aspect fractdem.201 as=aspect.201
d.his h=fractdem.201 i=aspect.201
```

More structured elevation models can be obtained by increasing the fractal dimension D given as a parameter `d`.

Line of sight. The line of sight analysis creates a viewshed for a specific point in an area based on the digital elevation model. The module `r.los` generates a raster map output with the cells that are visible from a user-specified

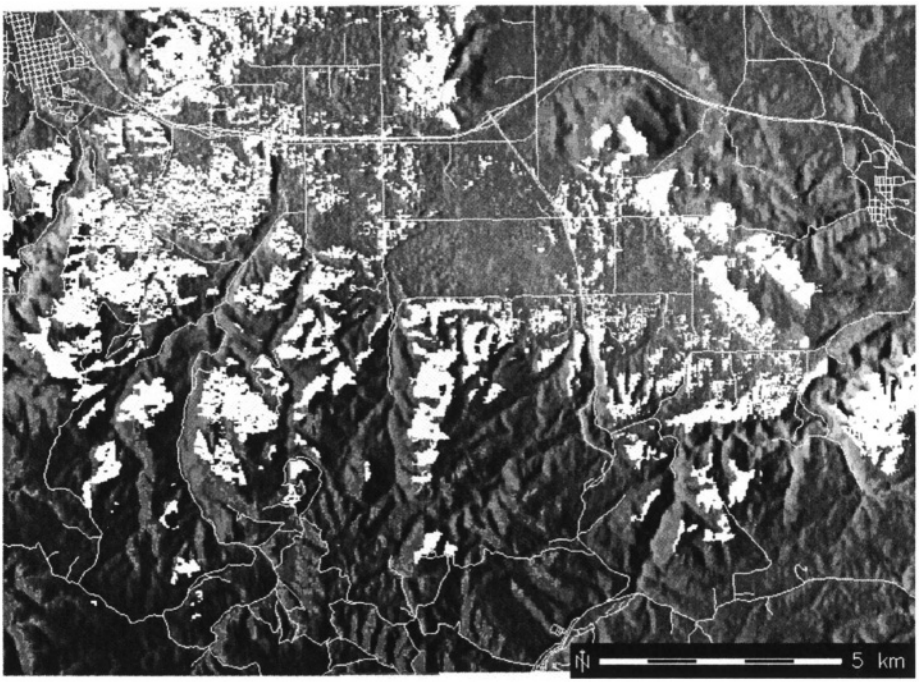


Figure 5.7. Visibility impact analysis of sample windpower plant east of Spearfish (cross northwest on the map, east of Spearfish). The visibility map is overlayed over the elevation model, and it is displayed together with the vector roads map

observer location at a given altitude over the ground. The output map cell values represent the vertical angle (in degree) required to see those cells from the observer location. We introduce the usage of this module with an example:

Suppose that there is a plan to build a wind power plant east of Spearfish at coordinates 593670E and 4926877N. Concerns about visual and noise impacts may arise for the residents in Spearfish, and our task is to find out where the 50 m high power plant will be visible. For the sake of simplicity, assume that sound propagates similarly to light. To perform this type of analysis, we can use a "line of sight" method implemented in the module `r.los`. Note that the module does not include any wind direction algorithm which may be needed to further analyze the noise distribution. We use the elevation model as an input for analysis, specify the coordinates and the height of the power plant. The `max` parameter is needed to define the maximum visibility distance (here 50 km):

```
r.los in=elevation.dem out=plant.los coord=593670,4926877\
                                                obs=50 max=50000
d.rast elevation.dem
d.rast -o roads
```

```
d.rast -o plant.los
d.barscale -m
```

The results show that the wind power plant is not visible in Spearfish, but widely in the other directions (compare Figure 5.7). Further impact studies would be required in the real world to identify the implications of planning a wind power plant. Potential wind power and other impacts have to be analyzed as well as other issues. Figure 5.8 shows a simplified planning procedure to find a location for a windpower plant. The wind power conditions may be coded in `r.mapcalc` to derive the final map from a set of input maps.

5.4.6 Landscape structure analysis and modeling

Because we could not cover every command and capability useful for processing and analyzing raster data, we would like to encourage you to further explore additional modules, such as `r.le` commands (Baker and Cai, 1992;

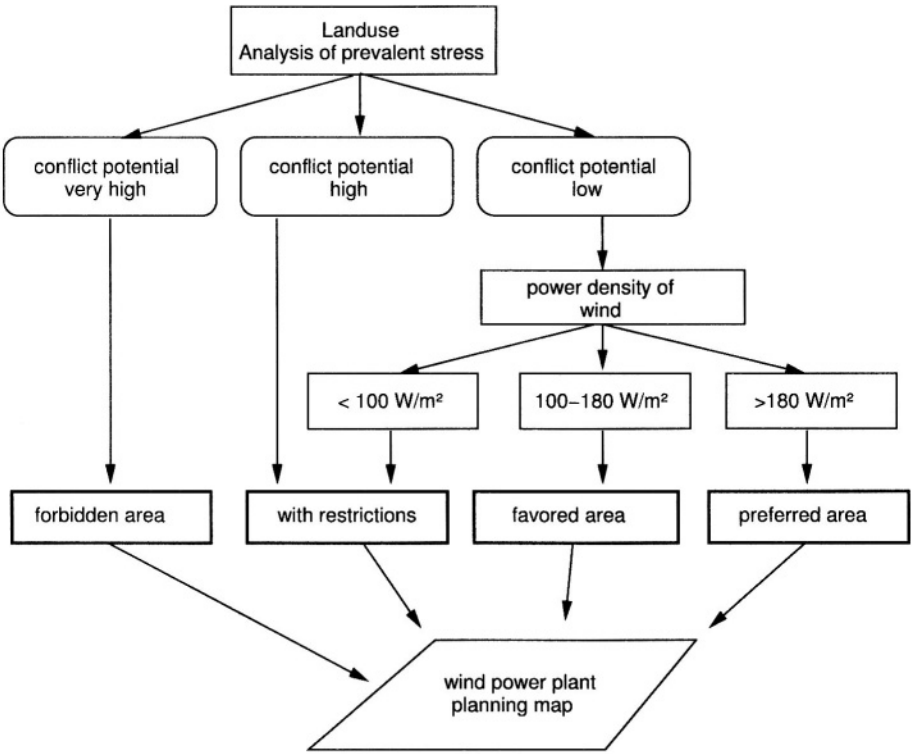


Figure 5.8. Simplified planning procedure to find a location for a windpower plant

Baker, 2001) for quantitative analysis of landscape structure: `r.le.patch`, `r.le.pixel`, `r.le.setup`, `r.le.trace`; `r.topmodel`, and `r.topidx` for hydrologic modeling, or `r.ros` for wildfire spread simulation. For complete list of modules see the GRASS users manual called by `g.manual`.

NOTES

1 Spearfish data set related documents, `http://grass.itc.it/data.html`

Chapter 6

WORKING WITH VECTOR DATA

The vector data model is used for representation of geographic phenomena as geometric objects composed of points, lines and areas. Lines are used for roads, railroads, streams, or utility networks, while areas can represent soil types, land use categories, lakes, or zoning in urban areas. Vector data are stored using their coordinates. In GRASS, the vector data model includes the description of topology.

At the time of writing this book, the vector data support is undergoing significant changes. This chapter reflects the GRASS 5.3 vector capabilities that are somewhat restricted. Improved tools with 3D vectors and direct DBMS support are under development for the GRASS 5.7 version.

For the description and import of vector data, please refer to Section 4.2.2.

6.1. DIGITIZING VECTOR DATA

A paper map can be converted to digital form by manual digitizing. In general, there are two ways to digitize a map:

- using a digitizing board or
- digitizing heads-up (on screen).

In the first case, the map is placed on the **digitizer board,** which provides a special digitizing mouse. The corners are selected by a mouse click and their respective coordinates are entered using the keyboard. This process is called *registering a map.* Then the lines and points on the map are digitized using a mouse. The advantage of this method is that the user always sees the entire map. However, the high cost of the equipment and the possibility that

the map could be shifted during the digitizing, if it is not properly mounted, are significant disadvantages. Furthermore, the paper map must be free of distortions to prevent displacements.

On-screen digitizing requires a scanned and geocoded raster map that is displayed in the GRASS monitor. All features will be digitized using the mouse. It is not necessary to register such a map because it is already geocoded. The advantage of this method is the possibility to zoom in and thus achieve an improved accuracy. Apart from an access to a scanner, no additional equipment is needed. The major disadvantage is the more difficult orientation on the map.

The following section deals only with heads-up digitizing using mouse. The installation of digitizing boards is described at the GRASS Web site.

6.1.1 General principles for digitizing topological data

To explain the digitizer module, we consider an example of vectorizing features from a scanned topographic map. We assume that the map was scanned, accurately geocoded and imported into GRASS in raster format (see Section 4.1.4 on geocoding scanned maps). Although it may be possible to automate the vectorization of a raster map using `r.line` or `r.poly` (see Section 5.3.1), problems often arise from overlapping lines, dots, map signatures etc. and manual digitizing is necessary.

There are few general recommendations for digitizing map features which can minimize the potential accuracy problems. The recommendations are mostly based on the fact that to make the map readable some features are exaggerated compared to their their size at a given scale:

- Line features should be digitized along their center-line, e.g. along the middle of a road. A line label point should be placed on the line;
- Area features should be digitized by following the center-line of area boundary lines. An area label point should be placed in the center of the area;
- Point features should be digitized at the center of the object, e.g. a point in the center of a map symbol representing the point feature or at the reference point of such a symbol;
- The points defining the line or polygon boundary should be selected at a density that is sufficient for preserving the geometry of the digitized features.

Rules for digitizing in topological GIS. When working in a topological GIS such as GRASS, following certain rules is recommended, in order to benefit from the topological features of the software. The following rules apply to the vector data (from *GRASS 5 Programmer's Tutorial,* Neteler, 2000):

- Arcs should not cross each other (i.e., arcs which would cross must be split at their intersection to form distinct arcs);
- Arcs which share nodes must end at exactly the same points (i.e., must be snapped together using the *snapping* function of the digitizing module). This is particularly important since nodes are not explicitly represented in the arc file, but only implicitly as end points of arcs;
- Common boundaries should appear only once (i.e., should not be double digitized);
- Areas must be explicitly closed. This means that it must be possible to complete each area by following one or more area edges that are connected by common nodes, and that such tracings result in closed areas;
- It is recommended that area features and linear features be placed in separate vector map layers. However, if area features and linear features must appear in one layer, common boundaries should be digitized only once. An area edge that is also a line (e.g., a road which is also a field boundary), should be digitized as an area edge (i.e., feature type A) to complete the area. The area feature should be labeled as an area (i.e., feature type A in the `dig_att` file). Additionally, the common boundary arc itself (i.e., the area edge which is also a line) should be labeled as a line (i.e., feature type L in the `dig_att` file) to identify it as a linear feature.

Now we explain the digitizing process in detail.

6.1.2 Digitizing in GRASS

Manual digitizing is done by running `v.digit` after a GRASS monitor has been started. First select the digitizing device. To use the mouse digitizer, choose "none" as digitizing tool. After specifying a new or existing vector map, you will get to the vector metadata page containing basic information and the map boundaries. In the case of using an existing map the boundary coordinates are set according to the map; in the case of a new map the current region boundary coordinates are defined. Besides the date, map creator and title, the map scale has to be entered. When working on a new map, we strongly recommend changing the default "Map's scale" from 1:1 to the correct scale of the map (e.g. 1:24,000). This value is important for the snapping tool as the snapping threshold is calculated from the map scale. A sample metadata screen may look like this:

```
Provide the following information:

Your organization     GRASS Development Team________
Todays date (mon, yr) 8/23/90 _______________
Your name             grass_________________
Map's name            Output from v.patch_____________________
Map's date            ___________
Map's scale           1:24000_____
Other info            ____________________________________________
Zone                  13____
West edge of area     587433.541 _____
South edge of area    4912505.24_____
East edge of area     611556.409_____
North edge of area    4929524.12 _____
```

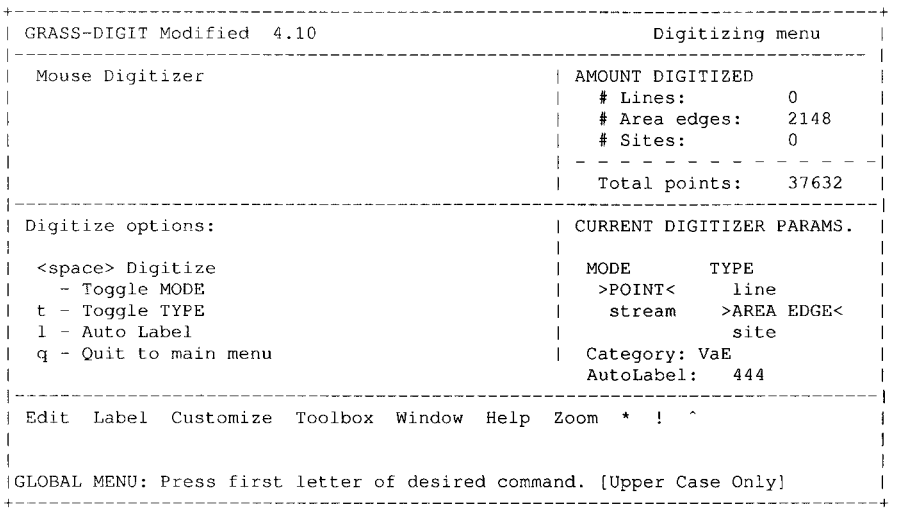
After completing this dialog, you will be transferred to the main menu of `v.digit`. The menus of `v.digit` are used as follows: To reach submenus from the main menu, enter the first (capitalized) letter of the menu name. A submenu is left with `q`. Other entries within the submenus are selected by entering the related key:

```
+-------------------------------------------------------------------+
| GRASS-DIGIT Modified 4.10                              Main menu  |
|-------------------------------------------------------------------|
| MAP INFORMATION                    | AMOUNT DIGITIZED             |
|   Name:        Output from v.patch |   # Lines:          0        |
|   Scale:       24000               |   # Area edges:     2148     |
|   Person:      grass               |   # Sites:          0        |
|   Dig. Thresh.: 0.0300 in.         |                              |
|   Map Thresh.: 18.288 meters       |   Total points:     37632    |
|                                    |                              |
|-------------------------------------------------------------------|
| OPTIONS:                           |                              |
|                                    |                              |
|     Digitizer:     Disabled        |                              |
|                                    |                              |
|                                    |                              |
|                                    |                              |
|                                    |                              |
|-------------------------------------------------------------------|
| Digitize  Edit  Label  Customize  Toolbox  Window  Help  Zoom  Quit  * ! ^
|                                                                   |
|GLOBAL MENU: Press first letter of desired command. [Upper Case Only]|
+-------------------------------------------------------------------+
```

First, we want to display the scanned topographic map in the GRASS monitor to use it as digitizing reference. For this choose the option "Select A Backdrop CELL Map" (`B`) from the "Customize" submenu (`C`). You can list the available raster maps by entering `list`. After selecting the raster map, it will be shown in the background. It is also possible to display a vector map. For this, choose "Select An Overlay Vector Map" (`O`). Leave the "Customize" submenu with `q` to reach the main menu. We are now ready to start the digitizing procedure.

Note that `v.digit` provides a context help system which shows the meaning of each menu entry (use `H` to reach it).

Digitizing vector sites, lines and areas. The digitizing is started by opening the Digitizing menu by pressing `D`. It is important to choose the appropriate object type (site, line or area) by pressing `t` ("toggle type"). If desired, switch on the "auto labeling" function. When activated, the category number and optionally also the category label are automatically assigned to the vectors. GRASS will query for both values. An alternate method to enter/modify these labels is using `v.support` (menu entry "Edit the category file" in interactive mode, find details in Section 6.2.2).



The digitizing process itself is started with <SPACE>. Now switch over to the GRASS monitor, and start to draw lines using the mouse by clicking on the points representing the line with the left mouse button. The mouse menu is shown in the terminal window, the buttons provide different functions such as drawing nodes and lines, removing the latest drawn node and finishing/omitting a line. This requires a bit of experience but you will quickly feel familiar with the concept. This way you can digitize line by line (or just vector sites) while getting the vector features automatically labeled by using the autolabel function. Labeled lines have a different color than the unlabeled lines. The labeling status can be checked in the submenu "Toolbox" (`T`), the labels themselves are displayed from the submenu "Window" (`W`).

Generally, we recommend making an extensive use of the "zoom" function which can improve the digitizing accuracy significantly. The function is available everywhere, while digitizing as well as with key `Z` from the main menu. The "pan" (panning, select while digitizing from the mouse buttons menu)

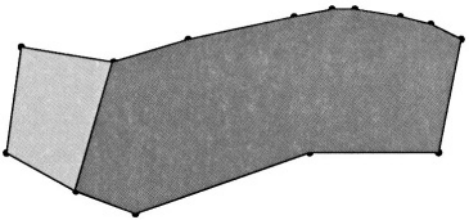


Figure 6.1. Digitizing common area boundaries in a topological GIS

function allows us to move the map into any direction without changing the zoom level.

When intersecting lines or connecting to lines, a node has to be inserted at the intersection. This is done by "breaking" the existing line which you want to cross or connect to. To break a line, enter the "Edit" menu (`E`) and select `b` ("Break a line"). Then you have to select the line to be broken and insert a node. You can now snap lines to this node which is explained next. In the same menu you also find functionality to move nodes, lines, to re-type a vector line (toggle between line and area type), to smooth a vector line with a spline, and to remove a block of lines.

When working with polygons, it is important to digitize common boundaries of adjacent areas only as a single line (see Figure 6.1). GRASS will automatically assign the common boundary to both areas. Never digitize a common area edge as two parallel lines! To verify, the "Toolbox" menu provides the function "Display Duplicate lines" (`d`) which highlights duplicate lines.

Snapping of nodes. Node snapping is necessary when digitizing a line which consists of several parts, or when closing a vector area. As mentioned above, closing areas is mandatory because only then is GRASS able to establish vector topology and assign label points. A great help is the built-in snapping function. When two nodes are close to each other (depending on the current snapping threshold), they will be moved into the same node and reduced to one node. When you receive a message that snapping cannot not be done, you have three options: you may either reject this area, zoom and vectorize again, or snap the open area in the "Edit" menu (`E`) using the "snap" (`s`) function. Such snapping problems are usually an indicator that you either have forgotten to change the map scale from 1:1 to the current map scale or, if it is correct, that you should zoom further on into the current map portion.

To optimize the *snapping* function, you can adjust the snapping threshold in the "Customize" menu using the `s` ("Set snapping threshold"). The snapping threshold should be chosen appropriately for the map scale. The value has to be entered in inches, but it will be converted to metric units based on the

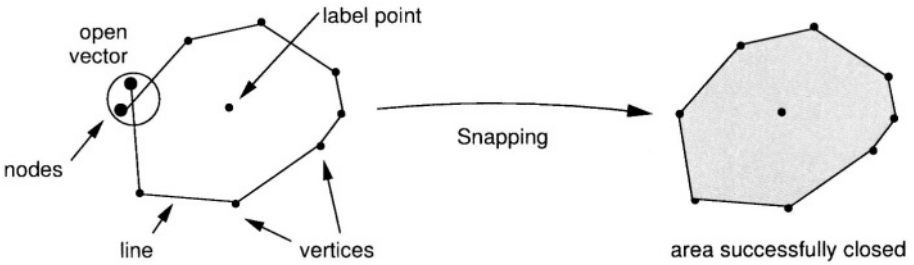


Figure 6.2. The node snapping function in GIS

projection units of the LOCATION. In general reasonable values for snapping thresholds depend on the map scale, such as:

- 1:5000 - 1:10000: Snap distance 1-2 m on ground
- 1:10000 - 1:25000: Snap distance 2-5 m on ground
- 1:25000 - 1:50000: Snap distance 5-10 m on ground

You should enter values around 0.002 for "Enter new Dig threshold" to fulfill the above recommendations. The threshold according to ground distances is calculated immediately. The module `v.digit` provides different colors for distinguishing various vector types and the label status. The color of snapped nodes will change, with open nodes being green by default, and snapped nodes turning to magenta.

Digitizing elevation contour lines. To make the digitizing of contour lines more efficient, GRASS offers a semi-automatic labeling of digitized contours. First, digitize the contour lines without attributes. Then switch to the "Label" menu by pressing `L`. Now hit `i` for "contour interval" and enter the appropriate value for interval of lines, default is the increment of 5 per line. The map units are usually meters, hence this represents 5 m contour intervals. The application of the values to a set of lines is done by digitizing a new temporal line across the contour lines for selection. This is done by mouse after pressing `c` for "Contour labels". Now select the lowest and the highest contour line. Both points are connected by a line crossing the contours in between. If one or both of these contour lines are not labeled yet, GRASS will ask for the elevation of this unlabeled line: "Enter elevation for this line". The labeling of the contours in-between will only be done if the defined elevation interval matches the values of the lowest and highest line.

Common digitizing problems and solutions. A common digitizing problem is that area boundaries are not closed or lines intended to be connected ("polylines") are broken. Lines too short and not reaching another traverse line are called "undershoot", lines too long which cross another line without having a node at the intersection are called "overshoot" (see Figure 6.3). To fix these problems, you should use the snapping function. Note that snapping is also implemented in additional vector modules such as `v.support`.

During the editing process within `v.digit`, the graphics screen can become cluttered for various reasons. Press the key ! to correctly replot the graphics.

You can find further information about digitizing in the *CERL-Tutorial: v.digit* which is available on the GRASS Web site (section "Documentation").

Post-Digitizing issues. As a general rule, you should always run `v.support` after using `v.digit` to build the topological information for the vector map. If needed, you can add/modify the category labels for vectors. You have to start `v.support` in interactive mode and select the option "Edit the category file". See Section 6.2.2 for details.

Maps containing intersecting vectors without nodes at the line intersections are called "spaghetti maps" which are topologically incorrect. To resolve this problem, the module `v.spag` can be used (see Figure 6.4 illustrating the functionality) which inserts nodes at the line intersections. To avoid unintended

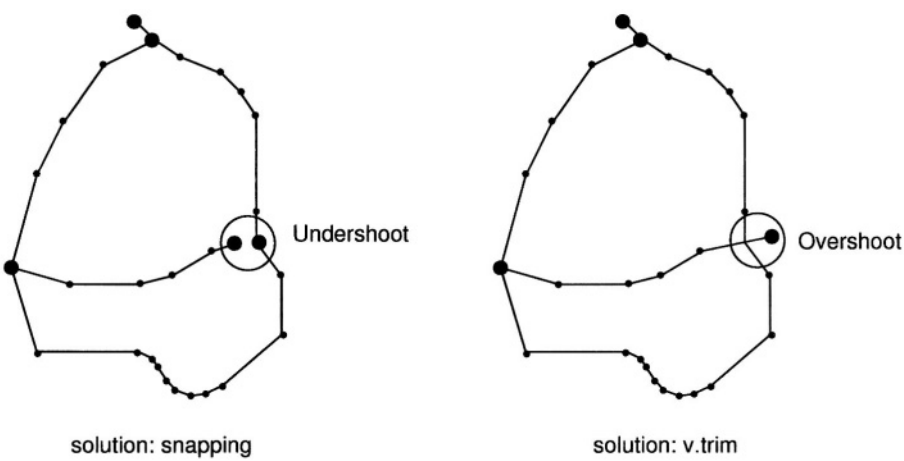


Figure 6.3. "Overshoots" and "undershoots" in vector maps

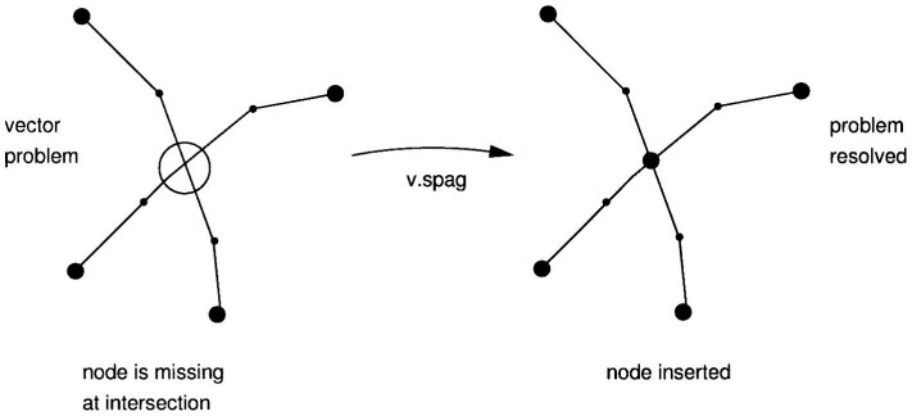


Figure 6.4. Correction of "spaghetti digitizing"

modifications of your digitized data, running it on a map copy (generate with `g.copy vect=oldmap, newmap`) is recommended.

Additional modules for cleaning the vector data are `v.clean` and `v.prune`. Both have to be used carefully. Again, we recommend working on a copy of the original vector map to avoid complications. The module `v.clean` removes "dead lines" from a vector map. "Dead lines" are vectors that have been marked deleted in `v.digit`. The module `v.prune` can be used to remove excessive nodes from a map. Use it with care as you can simplify complex areas/polygons to squares or triangles if you remove too many nodes.

6.2. METADATA AND ATTRIBUTES MANAGEMENT

Internally, GRASS 5.3 handles only one attribute per vector. However, by linking GRASS to an external database management system (DBMS), multiple attribute per vector object can be managed. The management of multiple attributes for vector data in conjunction with an external database management system will change substantially for GRASS 5.7 that is under development and is not covered in this book; please check the Web site for the most current updates. To find information about a related module that is already available, run (preferably in HTML mode):

```
g.manual database
```

6.2.1 Managing metadata of vector maps

To display metadata for a vector map, run

```
v.info map
```

The output includes a map title, production date, creator, vector level (topology present or not), number of categories, lines, areas and islands (areas in areas), projection, map boundary coordinates, map scale, and further comments.

Metadata modifications. Most metadata for vector maps can be managed with `v.digit`. After loading the vector map, the metadata screen allows us the modification of these data. First, make sure that map scale matches the true map scale, because some vector calculations and the snapping threshold depend on it. If the map scale is not correct, use `v.digit` to change it within this metadata screen. You can leave the metadata screen with <ESC><ENTER> if you don't intend to digitize the map. Then `v.digit` can be left either with answering "Shall we continue? [y]" with `n` or leave it from the main menu with `Q` (quit).

A timestamp representing the date of map generation or last modification can be applied to the vector map. Use the module `v.timestamp` to generate, query or delete a timestamp. Please refer to Section 5.1.4 to learn more about absolute and relative timestamps and time ranges.

6.2.2 Map attributes modifications

As we have already explained, vector lines and areas have assigned a category number (attribute ID) and an optional category label (attribute text). The internal vector ID is usually not visible to the user. Both category number and category label may be shared between vectors. For more details please read Section 4.2.1. Attributes can be modified by the module `v.support`. Besides building the topological information the module provides access to the category labels file, the internal attribute table. The module menus are a bit old-fashioned, but this may change in future.

You have to start `v.support` in interactive mode to reach the menus (omitting a parameter at startup). For modifying text labels, select the menu entry "Edit the category file". If no category labels are present in the map, the "Highest Category" in the next screen will be zero. In this case, enter the number of attributes you intend to apply to the map. Otherwise the "Highest Category" represents the highest category number appearing in the table. Leave this screen with <ESC><ENTER> to proceed to the table. Here you can modify individual entries using the cursor keys. To quickly move around in large category tables or to reach a certain attribute you can enter a line number in the bottom line of the window and press <ESC><ENTER> to go there. To leave it, either scroll to the last screen of the category table or enter `end` at the bot-

tom line of the window and press <ESC><ENTER> to leave the table. With <CTRL><C> you can interrupt the editing and leave the module with unsaved changes.

To check a map for unlabeled lines and areas, you can use `v.digit` ("Toolbox" and "Window" menu, see Section 6.1.2 for details).

If you have a map without category labels which you want to label with a unique value or incrementally, the modules `v.llabel` (for line and vector sites maps) and `v.alabel` (for area maps) are useful. They will apply labels to all unlabeled vector objects.

6.3. VIEWING AND ANALYSIS

First, we show how to display vector maps. Then, we explain how to perform geometrical operations, and how to modify vector map layers based on attribute selection.

6.3.1 Displaying vector map layers

As we have already done earlier, vector data can be displayed in the GRASS monitor window using the command `d.vect`. For example, to display the streams in the Spearfish region run:

```
d.vect streams col=blue
```

This will display the selected vector map in the monitor; in the case that another map is already present, it will be overlayed. If you want to display only selected vectors in a map, use the `catnum` parameter. The selection is done by category number given as a single value or a comma-separated list. If you don't define the color, the vector map will be displayed in white, so you need to have a non-white background or raster map in your monitor to see it.

To zoom within the map, you may use `d.zoom`. The module is controlled with the mouse buttons, the context menu is shown in the terminal window.

If you want to display vector areas as filled polygons, run `d.vect.area` instead. For example, a colored `soils` polygons vector map is shown with

```
d.vect.area -r soils
```

Optionally the border and polygon fill colors can be defined. The parameter allows us to limit the display areas to selected category numbers (use `v.report` to find out associated labels which is explained below).

To display category labels in the map, use `d.vect.labels`. An example:

```
d.vect.labels soils attr=string
```

Font size, colors, label marker etc. can be customized.

To see all or selected vector maps, the script `slide.show.sh` can be used. It only requires an open GRASS monitor. When running it with flag `-v`, it will show all available vector maps (without this flag it shows raster maps). Optionally, you can define a name prefix to see only selected maps.

To get a list of the maps currently displayed in the GRASS monitor, use `d.frame -l`.

Querying vector information. General information such as map title, creation date, scale, number of categories, lines and areas, and boundary coordinates from a vector map are provided by the command `v.info`. It also shows if the topology for a vector map has been built. If not, `v.support` should be run. To get a list of the vector map attributes (category numbers and labels), optionally with length or area sizes, use `v.report`:

```
v.info soils
v.report soils type=area units=h
```

The latter allows us to query line lengths and area sizes of vector features with parameter `units`. The map type, either line or area, has to be provided with parameter `type`.

The module `v.what` retrieves area information from polygons queried by a list of coordinates. As an example we query two points in the `soils` map of Spearfish:

```
v.what soils east_north=596954,4924870,591797,4926687
```

It reports the areas for the polygons which include the given coordinates and the area attributes (here soil names). To get coordinates from screen, use `d.where`.

You can interactively query vector data in a GRASS monitor with mouse. The module `d.what.vect` allows you to query vector objects in one or more map layers. As an example, we query the farm fields map and the soils map:

```
d.what.vect map=fields,soils
```

The mouse context menu is shown in the terminal window.

The alternate query module is `v.area` which retrieves area information (but no lines) by mouse click. Optionally, the desired area can be highlighted or filled in a different color.

6.3.2 Intersecting and clipping vector maps

Besides visually overlaying maps on the screen, you sometimes need to merge (intersect) two or more maps. The module `v.patch` provides this functionality. Unlike merging raster maps, intersection of vector maps is not a trivial task. Internally, new vectors have to be generated, because for each new

vector intersection the existing lines have to be broken and a new node has to be inserted (compare Figure 6.4). Please note that in GRASS 5.3, all vector modification tools ignore the current geographic region settings and always operate on the full map. The map boundaries are extracted from the vector file headers. As an example, we patch the soils map with the farm fields map (Spearfish region):

```
v.patch in=soils,fields out=soilsfarms.patch
v.info soilsfarms.patch
```

In the resulting intersection map `soilsfarms.patch`, the boundaries of the geographic region will be expanded to encompass the maximum geographic region as defined by the patched maps. The scale will be set equal to the smallest (i.e., most gross) scale used by any of the patched map layers. Remember that the scale will affect the node-snapping thresholds as implemented in `v.digit`, `v.support` and `v.spag`. The `v.info` command confirms that no topology is present yet. This has to be built with `v.support`:

```
v.spag -s soilsfarms.patch
v.support soilsfarms.patch
```

As there is no duplicate line removal in `v.patch`, the new map has to either be edited with `v.digit` or cleaned with `v.spag`. The latter must be done with care as explained above.

Clipping vector maps. To clip a vector map to modified map boundaries the script `v.cutregion.sh` is available. You first need to change your current region to the region of interest with `g.region` or `d.zoom`. Then you apply the script `v.cutregion.sh` which clips the vector map according to the current region settings and writes a new vector map.

A vector map intersecting tool which takes into account the shape of polygons is `v.cutter`. This module generates a new map based on an intersection of two input maps, `cutter` and `data`. The map `cutter` determines the "active" areas for the input map while `data` delivers the area contents. The attributes of created polygons will be generated from the attributes of the `data` map. As an example, we cut out the soils only for areas present in the map `rstrct.areas`:

```
d.vect.area -r rstrct.areas
v.cutter cutter=rstrct.areas data=soils out=soils.cut type=area
v.spag -i soils.cut
v.support soils.cut
v.info soils.cut
d.vect.area -r soils.cut
d.vect.labels soils.cut size=7 col=red
```

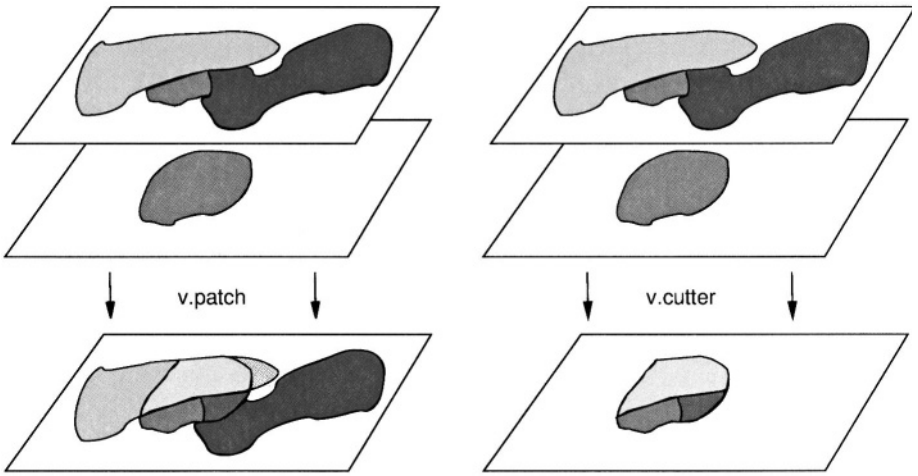


Figure 6.5. Possible results of intersecting vector data using `v.patch` or `v.cutter`

The new map contains soil information only for the restricted areas. This is also a convenient way to create masks based on vector maps.

To mask vector maps based on this method, you may digitize a mask area with `v.digit` or generate it with other tools. The mask area should be labeled with a category number (e.g. 1). The vector areas will determine the outline of the final vector map. The module `v.cutter` produces a new map that contains all the vector data from the `data` map that fall into the extent of the vector mask map given as map `cutter`. Figure 6.5 illustrates the difference between `v.patch` and `v.cutter`.

6.3.3 Map reclassification

Vector maps can be reclassed in a similar way as raster maps. The module `v.reclass` reclassifies vectors according to user defined reclass rules. The module is used in a similar way as `r.reclass`. As an example, we can reclass the `fields` map into two categories: private and "Black Hills National Forest". The ASCII table containing the reclass rules can be written with any text editor. It will contain:

```
1 thru 62 = 1 private
63        = 2 Black Hills Natl. Forest
```

This table will be applied to the map to generate a new reclassified vector map. The module `v.report` can then be used to get the list of input categories:

```
v.report fields type=area
cat fields.recl | v.reclass fields type=area out=fields.recl
```

```
v.support fields.recl op=build
v.info fields.recl
d.vect fields.recl
d.vect.labels fields.recl
```

The new map contains only two categories. With flag `-d` you can dissolve common boundaries of adjacent polygons with same category assigned during the reclassification. Note that there are parallel lines in the Spearfish `fields` map which represent roads.

6.3.4 Feature extraction from vector data

To extract vector objects from a vector map, you can run `v.extract` with the desired category(ies) listed by the parameter `list`. This will extract the selected vectors into a new map. As an example, we can extract the fields owned by C. Mitchell into a new vector map. We can get the category numbers of the vector polygons from `v.report`:

```
v.report fields type=area
v.extract fields type=area out=fields.mitchell new=0 list=10-15
d.vect.area -r fields.mitchell
```

The parameter `new` is set to zero to keep original categories, optionally, a new category number can be specified. For a larger list, the selection can be written into a file, and its name is then specified as the parameter `file`. The new vector map contains only the selected areas. You can dissolve common boundaries when adjacent polygons have the same category by using the command with the flag `-d`.

6.4. VECTOR DATA TRANSFORMATIONS TO/FROM RASTER AND SITES

First, we explain the transformation of raster and site data to the vector data model, then we show how to change vector data to rasters. Depending on the type of geographic phenomenon that the vector data represent, we distinguish two types of transformation from vector data to raster:

- for geometric features (points, lines, areas) we use direct transformation from vectors to raster lines/areas or sites;
- for continuous fields (isolines, contours) we need spatial interpolation to transform from vector lines to rasters;
- feature extraction from raster objects to vectors requires vectorization.

Figure 6.6 shows an overview of available conversion techniques.

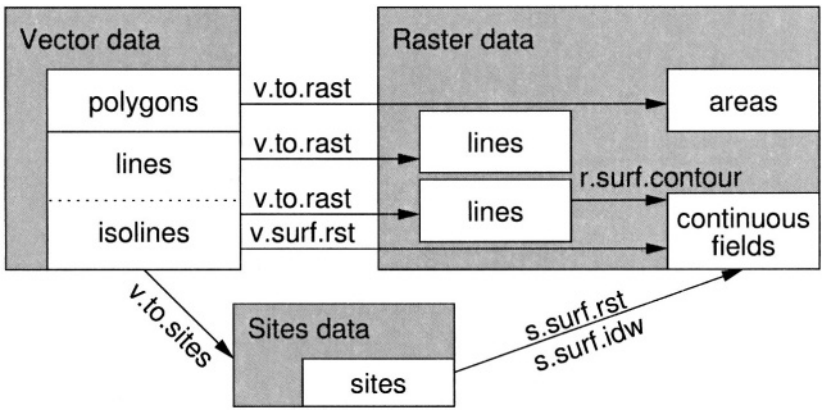


Figure 6.6. Methods for transforming and interpolating vector data to raster and site data

6.4.1 Automatic vectorization of raster data

Sometimes we need to convert existing raster data (lines, areas) to vector data. This can either be done by manually digitizing in `v.digit` or by an automated procedure, described below. GRASS provides modules for the following types of automated vectorizing:

- vector lines: `r.line`
- vector areas: `r.poly`
- vector isolines: `r.contour`
- convex hull: `s.hull`

The first three commands are explained in Section 5.3.1, along with some examples.

To generate an outer convex boundary for a set of points, the *convex hull,* we can use the `s.hull` module. A raster or a vector map has to be transformed to the sites using `r.to.sites`, and `v.to.sites` respectively, before the module can be applied. As an example, we extract all polygons labeled with soil type "VaB" (Vale silt loam) from the Spearfish soils raster map and generate the convex hull map surrounding the area where this soil type appears:

```
r.report soils
r.mapcalc "soils.VaB=if(soils == 51, 51, null())"
d.rast soils.VaB
r.to.sites -a soils.VaB output=soils.VaB
s.hull -s soils.VaB vect=soils.VaB.hull
d.vect soils.VaB.hull col=blue
```

The resulting vector area contains all areas where Vale silt loam occurs.

6.4.2 Direct transformation of vector data to raster or sites

The direct transformation of vector maps into raster lines or areas requires that all vectors are labeled. All unlabeled vectors will be omitted and will thus not appear in the output raster map. Labeling of vector maps is explained in Section 6.2.2.

The module `v.to.rast` generates a raster map from an input vector map. Transformation of vector objects to raster cells depends on the current resolution. The resolution settings can be defined with `g.region` using the `res` (resolution) parameter. You may try several resolutions to see the effect.

If you want to get only the vector area boundaries in a raster map, you can convert the vector area map to a vector line map with `v.area2line` (losing the original labels), then label the lines again with `v.llabel` and convert to raster with `v.to.rast`. An example:

```
v.area2line soils
v.llabel -i soils.2
g.region -p res=30
v.to.rast soils.2 out=soils.borders
```

It is important to define the target raster resolution before converting the vector map to raster. The resulting raster map `soils.borders` contains only the outlines of the soils areas.

To generate a sites map from vector data `v.to.sites` can be used. It transforms vector lines to sites. Without the flags `-ai`, only vector point data are transformed; using these flags, points defining the vector line segments are used. The density of the points along the line is controlled with `dmax` parameter. If these points are farther apart than the `dmax`, additional points are interpolated along the vector lines. As an example we interpolate sites along the roads map in Spearfish:

```
v.to.sites -ai roads out=roads dmax=500
```

This generates a sites map with points along the road at a maximum distance of 500 m (note that it is not the minimum distance!). If the present vector nodes in the input map are further apart than 500 m, new points are interpolated.

6.4.3 Interpolating raster surfaces from contour lines

GRASS provides several different methods for interpolation of raster surfaces from vector data (Figure 6.7). Depending on the method, the surface can be interpolated directly from vector data, or the vector data must first be transformed to raster lines and interpolated with the related raster module, or to sites and interpolated with sites interpolation tool. The following modules can be used:

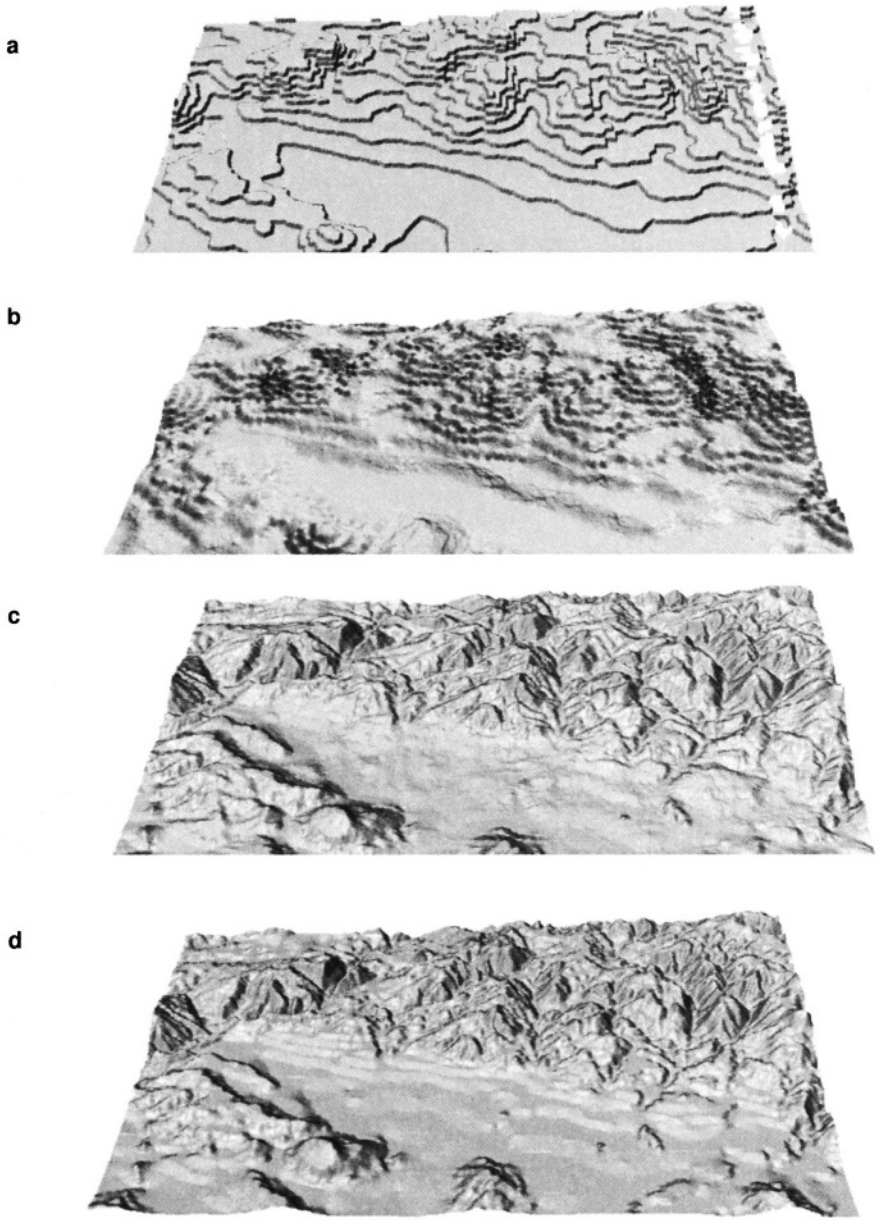


Figure 6.7. Interpolation of a raster elevation map layer from vector data (contours) a) voronoi polygons, b) IDW, c) `v.surf.rst` with default parameters, d) `r.surf.contour`. Note that because of their limitations, the first two methods were applied to a smaller data set than the examples c) and d)

- `s.voronoi` can be used in special cases when only the values assigned to contours should be in the resulting raster, leading to a discontinuous surface (see Figure 6.7a). It requires transformation to sites and further transformation of the voronoi polygons vector map to raster, as described in the Section 7.3.1;

- `s.surf.idw` is based on *inverse distance weighted* interpolation (IDW) and can be used after transforming the contours to sites (see Section 7.3.1 and Appendix B);

- `v.surf.rst` interpolates the raster directly from the vector data using the RST method: *regularized spline with tension* method (see Section 7.3.2 and Appendix B);

- `r.surf.contour` requires conversion of contours to raster lines and then linearly interpolates between contour lines.

The interpolation method should be selected based on the application. As with all transformations to raster, we have to define the GRID RESOLUTION for the new raster map with `g.region` before applying the interpolation module.

RST interpolation method. The RST (Regularized Spline with Tension) method is explained in detail in Section 7.3. It has been adapted for direct input of vector isoline maps in the module `v.surf.rst`. Internally, all lines are converted to sites and interpolated using the same RST method as in `s.surf.rst`. The module provides numerous parameters, which you can use to tune the behavior of the interpolation function, calculate slope, aspect, and curvatures, as well as compute deviation and quadtree maps. For details please refer to the manual page; here we show a simple example (Figure 6.7c) We will interpolate a DEM using the vector contour map generated the previous section:

```
g.region res=30
v.surf.rst contour100 elev=elev100.rst &
d.rast elev100.rst
```

If elevation values are stored as category labels and not as category numbers, the flag `-c` has to be used. The module gives recommendations for optimizing the calculation depending on the input data, for example, by giving warnings about possible overshoots (interpolated values exceeding the range of given values over 15%) when an increase in tension and/or smoothing is needed. For further explanation of parameters and functionality of this module read the Section 7.3.2. A method to analyze the quality of interpolated maps is shown in Section 13.2.1.

Note that instead of digitizing, when the points defining the contour line were selected manually, most contours are nowadays generated automatically. Scanning of contours or their computation from a dense TIN or a raster leads to a very high density of points along the lines while there may be large areas between contours (especially in flat terrain) without any data. Such a representation of a surface with strongly heterogeneous spatial distribution of data points presents substantial challenge for most interpolation methods, which tend to create waves or steps along the isolines. Reducing the number of points on the lines (for example, by increasing the `dmin` parameter in `v.surf.rst`), adding points between the contours, and changing the interpolation parameters (e.g. lowering tension for `v.surf.rst` or increasing the number of points for IDW) helps to minimize the problem.

When there are large areas with sparse contours, rectangular segments may become visible, especially in the aspect map. While the error in the elevations due to the segments is usually negligible, it is not acceptable for shaded maps (see Section 8.1.2). The problem can be eliminated using two step interpolation. First interpolate the surface using `v.surf.rst`. If segments are visible, generate additional points sparsely but homogeneously distributed over the elevation surface using `r.random`. Transform the contours to sites using `v.to.sites` and merge with the sites file generated by `r.random`, e.g. using UNIX command `cat`. Finally, interpolate this merged sites file using `s.surf.rst`. The surface should be without segments.

Linear interpolation between contours. The module `r.surf.contour` requires the vector map to be converted to a raster lines map. The vector-to-raster conversion is done with `v.to.rast`. To interpolate the raster surface, run `r.surf.contour` with the name of the raster lines file and a name for the resulting raster surface. In our example (Figure 6.7d), we use the vector contour map generated in the previous section:

```
g.region res=30
v.to.rast contour100 out=contour100
r.surf.contour contour100 out=elev100
d.rast elev100
```

The module interpolates the elevation at a given cell from the uphill and downhill contour values by the true distance. To obtain good results it is important that the contour lines extend to the edge of the current region and there are no disjointed contour lines. Since a flood fill algorithm is used, running time grows exponentially with the distance between contour lines.

Chapter 7

WORKING WITH SITE DATA

Observed values or properties can be spatially referenced to a single point or, in GRASS terminology, a *site*. Site data represent either a discrete feature at a given scale, such as a city, an archaeological site or a hospital, or they are discrete samples of continuous fields such as data from climatic stations, measured elevation points, or bore-hole data. GRASS provides tools for management and analysis of sites map layers, as well as their transformation to vector or raster data. If the site data represent a continuous field, transformation to raster representation of this field is performed by spatial interpolation.

7.1. CREATING SITE DATA

We have described the GRASS sites data format and various ways of importing it from external sources in Section 4.3. In this section, we describe the methods and tools for creating new GRASS sites map layers. One approach is to modify an ASCII file exported from a database or other source using a text editor or `awk` (see example Appendix A). The file then should be copied to the `site_lists/` directory in your MAPSET. If this directory does not exist, go to your MAPSET directory and create it using `mkdir site_lists/`.

7.1.1 Digitizing site data

Point data can be manually digitized from a map and assigned an associated attribute using the `v.digit` module (see Section 6.1.2). Result will be a vector map layer with point data. At the beginning of a digitizing session, you should enter a new file name and correct the value of "map scale" in the following menu from 1:1 to your map scale. Because the digitizing is done from a map, a vector or a raster map can be displayed in the background

(C – Customize, O – backdrop vector map or B – backdrop raster map). The D key gives you access to the digitizing menu, t changes the vector type from lines to site. By pressing <SPACE> you can start to digitize single vector sites. The next step is assigning labels with l; for example, in the case of an archaeological site you can add the name to each site. When labeling, each vector site is assigned a category number with the text label as category label. For digitizing, mark the vector site in the map appropriately. A labeled site mark changes its color.

As mentioned before, the digitized sites are saved as point vector data and can be converted into GRASS sites model using v.to.sites. Manual digitizing of point data can be a time consuming process prone to errors; therefore, it is used only as a "last resort" solution.

7.1.2 Generating site data within GRASS

Several GRASS modules can be used to create a new sites map layer. For example, a raster can be transformed directly to sites using r.to.sites, which creates a site for each grid cell center expressed as east|north|#value. If the current resolution is lower than the resolution of the raster file, the nearest neighbor cell value is used. Optionally, the values stored in the raster map layer can be expressed as a floating point attribute or a third dimension using the -a or -z flags, as illustrated by the following examples showing the command (stored as a desc entry in the site file header) and the resulting sites file format (note that you can use the same name for your site and raster maps, because GRASS stores them in a different directory):

```
name|roads
desc|r.to.sites input=roads output=roads label=rd.category
form|||#
labels|Easting|Northing|#rd.category
605685|4927985|#3

name|slope
desc|r.to.sites -a input=slope output=slope label=slope
form|||%
labels|Easting|Northing|%slope
590235|4921965|%13.61707306

name|elevation.dem
desc|r.to.sites -z input=elevation.dem output=elevation.dem ...
form||||
labels|Easting|Northing|elevation|
590235|4921965|1193|
```

Similarly, a 3D raster volume can be transformed to 3D sites by using r3.to.sites, which is useful for viewing the 3D data with nviz (see Section 8.2.4).

A raster map layer can be sampled randomly using `r.random` resulting in a sites map layer which contains points that are the centers of randomly selected grid cells. For example, you can create a random grid points site file from our Spearfish DEM by running:

```
g.region rast=elevation.dem
r.random elevation.dem nsites=3000 sites_out=elev.rnd.3k
```

The module writes the coordinates of 3000 points along with the values from the `elevation.dem` as a decimal attribute to the site file `elev.rnd.3k`. You can specify the number of random sites to be generated either as a positive integer, or as a percentage of the raster map layer's cells (e.g., 10%). If you already have a site file and you want to add an attribute based on the values stored in a raster map, you can use `s.sample`:

```
s.sample archsites out=archsites.slope rast=slope
```

The new site file will now include the slope at each archeological site stored as a floating point attribute:

```
name|archsites.slope
desc|slope[rast] sampled at archsites[sites] by Nearest Cell
593493|4914730|#1 %5.75962543
```

Vector data can be transformed to sites using `v.to.sites`. You can transform only the point features (nodes) from your vector map layer, or all vertices that define the vector lines. If the distance between any two vertices on a line is greater than a distance given by `dmax` parameter, additional points are interpolated on the line using a spline function to keep the maximum distance between the vertices within the `dmax` range.

It is also possible to create new site data from an existing sites map layer by perturbing (adding a variable spatial deviation) the east and north coordinates using `s.perturb`. This deviation can be a uniform value or a delta value with normal distribution. Randomly distributed sites can be generated by `s.random`. Unlike the result of `r.random`, these sites will include only the coordinates and no attributes.

Generating site lists using UNIX pipes. Several GRASS modules can be used to produce output in a format suitable for input to `s.in.ascii`. For example, you can pipe output produced by `d.where` into `s.in.ascii` to create a site list file containing site locations selected by a mouse:

```
d.where | s.in.ascii sites=mysites
```

In another example, you can calculate distances of given points, stored as GRASS sites, to the nearest vector line. Here, the distances of archaeological sites to the closest road, given in a vector map, are calculated. The individual distances are stored as a new sites map (columns `east, north, sitesID, distance, vectorline-catnum`):

```
s.out.ascii -d archsites | v.distance roads |s.in.ascii fs='|'\
                                                    sites=dist
```

In this case, the field separator is required as `v.distance` outputs the pipe character.

7.2. VIEWING AND MANAGING SITE DATA

Basic information about a sites map layer can be obtained by running `s.info`. The output prints out the number of sites within the given region, number of dimensions, as well as the minimum and maximum values for coordinates, categories and attributes:

```
s.info archsites
```

If the output is not what you have expected, check your format – you can read your site data by going into the `site_lists` directory in your MAPSET and edit the file `archsites` using any of the UNIX text tools (see Appendix A) or any text editor.

7.2.1 Displaying site data and creating subsets

You can view your site data in the GRASS graphical window using `d.sites`, for example:

```
d.sites archsites color=red size=2 type=diamond
```

will draw your sites using a diamond symbol with 2 pixel size in red color. Use `d.site.labels` to label the sites using the category, floating point or text (string) attribute. You can also query your sites interactively, using the mouse to get the coordinate values and attributes of site(s) nearest to the location "+" selected using the mouse:

```
d.sites archsites
d.what.sites
[...]
"+" at 591582.609375(E) 4925092.15625(N)
archsites in PERMANENT  591583|4925280 24  "Hanson Ranch"
  Distance from "+":    187.84
```

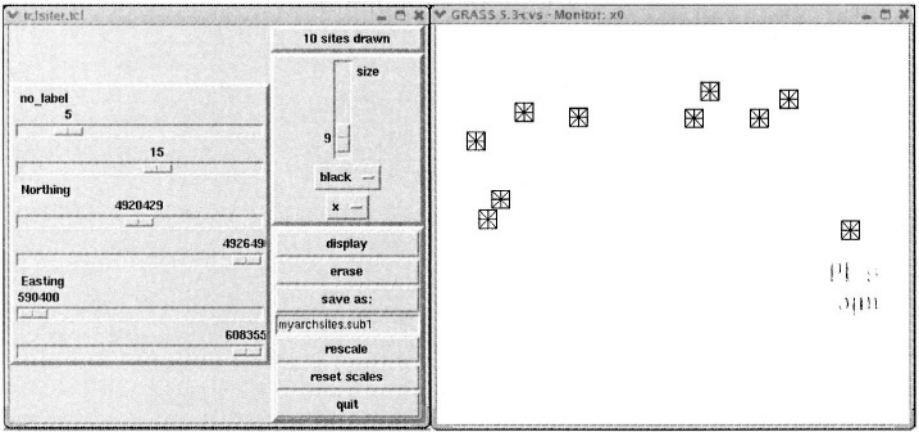


Figure 7.1. Selecting a subset `myarchsites.sub1` of site data `myarchsites` using `d.siter`, with categories 5-15 and northing > 4920429

To select subsets of site data with given coordinates, attributes, or categories use `d.sites.qual`. Its TclTk version `d.siter` allows you to display a subset of site data interactively, based on selected ranges of coordinates and/or attribute values. An example of using `d.siter` is in the Figure 7.1. Subsets defined by selected ranges of attributes may be visualized using various marker sizes, shapes, and colors. Subsets may also be saved to a new sites file. The command line version can be illustrated by the following example. It extracts a subset of sites from the map `archsites` with category numbers 5-15 that are located north of the given y (northing) coordinate:

```
d.sites.qual archsites rules=C1.5-15,D2.ge.4920429 \
             out=archsites.sub2
```

The subset is stored in a new site file called `archsites.sub2a`. You can find more examples of the rules syntax in the `d.sites.qual` manual page.

You can also use `nviz` to view your site data draped over a surface or in their 3D position (see Chapter 8).

To create a subset of sites located within a given subregion (defined for example by `d.zoom`), you can use the module `s.mask` with any raster with non-NULL cells:

```
d.zoom
[...]
s.mask archsites out=archsites.zoom raster=owner
Total: 25, Output: 8
```

The module finds eight sites in the subregion and writes them into the new sites file. You can use the same command to find which sites are located within the 30 m cells representing roads:

```
s.mask archsites out=archsites.roads raster=roads
Total: 25, Output: 2
```

7.2.2 Computing basic statistics

Simple statistical analysis of site data can be performed directly with several GRASS modules. For more sophisticated spatial statistics and geostatistics tools it is recommended to take the advantage of the bridge between GRASS and R, as described in Chapter 13, as well as by Bivand, 2000, or a link with gstat described in the same chapter.

Basic univariate statistics can be computed using s.univar. For example, for a random elevation data elev.rnd.3k created in the Section 7.1.2 you get:

```
s.univar elev.rnd.3k
[...]
        number of points 3000
                    mean 1570.4
      standard deviation 128.011
coefficient of variation 8.15147
                skewness -0.33675
                kurtosis -0.784016
         mean of squares 2.48253e+06
 mean of absolute values 1570.4
                 minimum 1186
           first quartile 1464.5
                  median 1592
           third quartile 1674
                 maximum 1836
```

You can run r.univar elevation.dem to compare the result with the basic statistical measures of the original data. Please refer to Appendix B for a list of basic statistical equations used to compute these measures. Further statistical analysis can be performed using additional GRASS sites commands, such as s.normal which supports computation of 15 different normality tests for a selected site attribute. The module s.qcount provides a test for complete spatial randomness using a quadrat method, while a sample semivariogram can be computed by s.sv. You can fit a semivariogram model to the sample semivariogram using the module m.svfit. However, these modules haven't been thoroughly tested, so they should be used with caution.

Note that besides using R and gstat, you can always transform your site data to raster or vector representation and use the raster and vector modules to greatly expand the possibilities for the point data analysis.

7.3. TRANSFORMATION FROM SITES TO RASTERS AND SPATIAL INTERPOLATION

Depending on the type of data, the transformation of site data to the raster data model can be performed using two approaches (Figure 7.2):

- for discrete phenomena, the transformation of site data to raster is done with `s.to.rast`. It creates an output raster file with the selected site attribute value in a cell where the site is located, while inserting NULLs elsewhere. If more than one site falls into one raster cell, the module will continue to import the sites and the last imported site value will determine the cell value;

- if the site data represent sampling points of a continuous phenomenon, such as elevation, temperature, or chemical concentration, raster representation of this field should be computed by spatial interpolation.

For 3D sites, the discrete transformation to a 3D raster volume is performed using `s.to.rast3` and the spatial interpolation is supported by trivariate versions of the available interpolation modules, as described in the following sections.

7.3.1 Selecting an interpolation method

Spatial interpolation transforms site data to the raster representation using a function which passes through (or close to) the given sites. Because there exists an infinite number of functions which fulfill this requirement, additional conditions have to be imposed, leading to a number of different interpolation techniques. GRASS offers interpolation functions which are based on conditions of locality (voronoi polygons, inverse distance weighted) and smoothness (splines). The methods based on geostatistical concepts can be applied by taking advantage of the link with the Open Source geostatistical tools (see Chapter 13). It is important to keep in mind that different methods, and often even the same method with different parameters, can produce quite different surfaces (see, for example, Figures 7.3, 7.4, 7.6). A good knowledge of the modeled phenomenon is needed to evaluate which one is closest to reality. Statistical measures of accuracy do not always ensure that the properties of the interpolated surface are adequate representation of the behavior of the modeled phenomenon.

Before interpolating in GRASS, it is necessary to set the resolution of the resulting map layer with `g.region` (see Section 4.1.2). Several modules can then be used to interpolate a raster map from scattered site data.

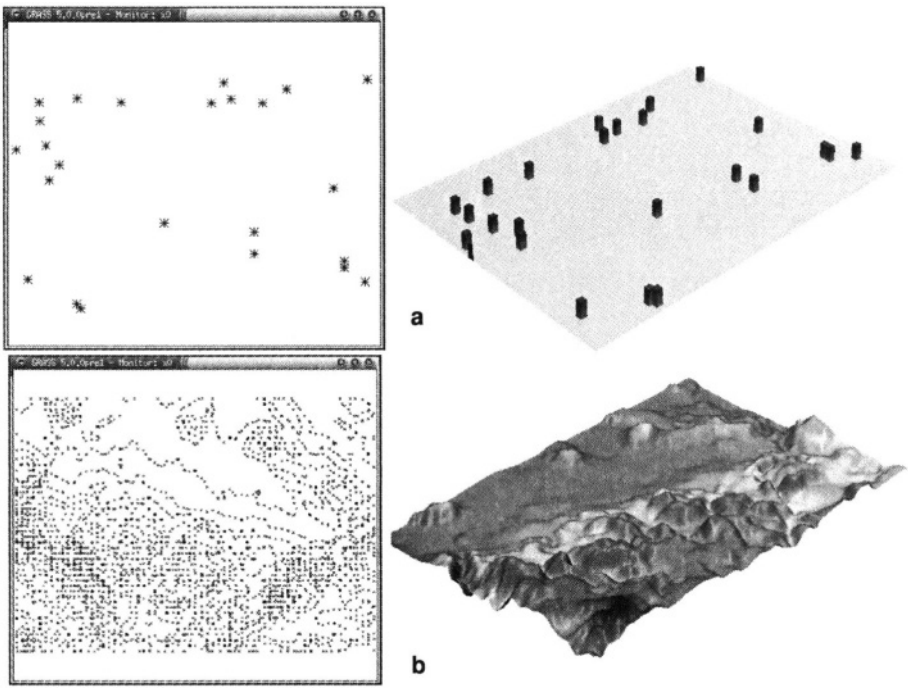


Figure 7.2. Conversion of site data to raster for: a) discrete phenomenon – archaeological sites; b) continuous phenomenon – elevation

Voronoi polygons. This method is suitable for transformation of qualitative site data when the condition of continuity is not appropriate. The site attribute is simply assigned to all cells within its natural neighborhood defined by a voronoi polygon (Fortune, 1987). The module `s.voronoi` generates these polygons as a vector map layer with each polygon carrying the site attribute. You can then transform this vector map layer to a raster using `v.to.rast`, resulting in a surface composed of discontinuous, horizontal patches (see Figure 7.3 a). The procedure, using the random samples of Spearfish elevation data (see Section 7.1.2) starts with modification of the `elev.rnd.3k` site file in the `site_lists` directory in your MAPSET:

```
sed 's/%/\#/' elev.rnd.3k > elev.rnd.3ki
s.voronoi elev.rnd.3ki vect=elev.rnd.3k
v.support -r elev.rnd.3k
v.to.rast elev.rnd.3k out=elev.rnd.vor
r.colors elev.rnd.vor rast=elevation.dem
```

The module `s.voronoi` expects the values as category numbers, therefore we have used the `sed` command to change the prefix % to #. The resulting

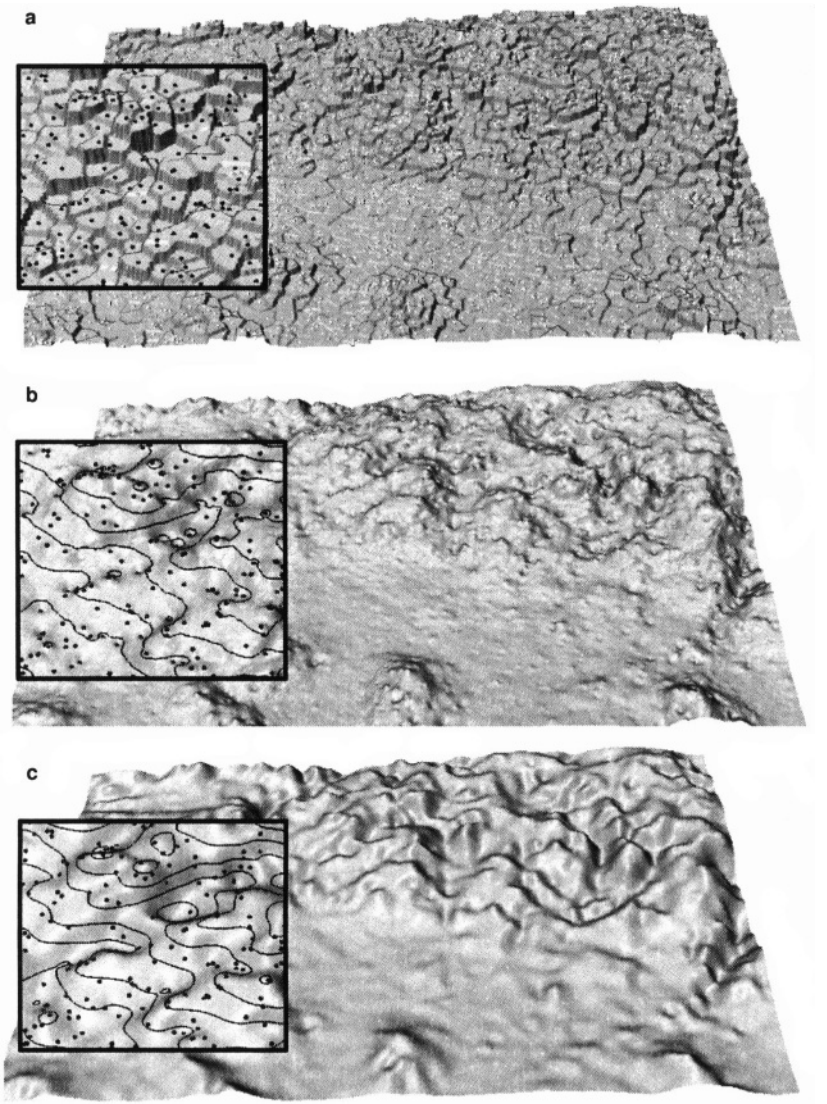


Figure 7.3. Interpolation methods available in GRASS and the resulting surfaces: a) `s.voronoi`, b) `s.surf.idw`, note the small peaks and pits around the data points, c) `s.surf.rst`. The surfaces are interpolated from the 3000 random samples of the Spearfish 30 m DEM

raster depends on the spatial distribution of input data. For example, the raster generated from contour points is quite different from the one generated from randomly distributed samples of the same surface (compare Figure 7.3a and Figure 6.7a). These figures clearly demonstrate that voronoi polygons are not a good choice for continuous fields, but they may be appropriate for numerous applications in ecosystem studies or geomarketing.

Inverse distance weighted average (IDW). This approach calculates the value for each grid point as a weighted average of values at the *n* closest sites (Burrough and McDonnell, 1998, see the equation in Appendix B.8). In the GRASS module `s.surf.idw` weights are inversely proportional to a power $p = 2$ of distance and the default $n = 12$. It is a simple approach, however, the results are less accurate compared to other methods such as splines, kriging or multiquadrics (Mitas and Mitasova, 1999). Often the method does not reproduce the local shape implied by data and produces local extrema at the data points (Figure 7.3b, also noticeable as small circular contours around the given points). The module is useful for rough interpolation of smaller data sets, especially at lower resolutions, when the density of points is higher than the density of the resulting grid points. The Figure 7.3b was created by:

```
s.surf.idw elev.rnd.3k out=elev.rnd.idw
```

Regularized Spline with Tension (RST). The method computes the values at grid points using a function which simulates a thin flexible plate passing through or close to the data points (Figure 7.3c). It is the most general and accurate method available in GRASS but it may require tuning of parameters to achieve optimal accuracy. Optionally, it also computes topographic parameters and partial derivatives of the modeled surface (see Chapter 12). The bivariate (2D) version is called `s.surf.rst` and the trivariate (3D) version is `s.vol.rst`. There is also a quad-variate experimental version available (e.g., for 3 spatial dimensions and time) called `s.volt.rst` for those who are interested in development of multivariate interpolation capabilities. The method, its properties and examples are described in more detail in the following sections.

7.3.2 Interpolating with RST: tuning the parameters

Bivariate Regularized Spline with Tension (RST, Mitasova and Mitas, 1993, Mitasova et al., 1995, Mitas and Mitasova, 1999) is implemented in GRASS as `s.surf.rst`. To interpolate the Spearfish elevation random sites `elev.rnd.3k` (generated in Section 7.1.2) to a raster map layer `elevrnd3k.def`, we can simply run the module with its default settings (Figure 7.3c):

```
g.region res=30 -p
s.surf.rst elev.rnd.3k elev=elevrnd3k.def
```

While the results may be satisfactory for many applications, it is worth exploring the full functionality of this module because it provides a number of additional capabilities, ranging from tuning the character of the resulting surface to computation of topographic parameters. Here, we discuss how to choose the parameters to optimize the spatial interpolation and evaluate its accuracy. In Chapter 12, we demonstrate the use of `s.surf.rst` for topographic (surface geometry) analysis.

To take the full advantage of the `s.surf.rst`, understanding the principles behind the method is important. The mathematical description is given in the Appendix B; here we provide only a verbal description with illustrations. The RST function minimizes a specific measure of surface smoothness (also called smoothness seminorm or roughness penalty) and simulates a flexible sheet forced to pass through the data points while minimizing its energy (Mitas and Mitasova, 1999, Wahba, 1990, Talmi and Gilat, 1977). Properties of this function can be controlled by the *tension* and *smoothing* parameters. See the difference in the impact of these two parameters illustrated by two animations at the Multidimensional Spatial Interpolation Web site[1].

Tension parameter. Tension tunes the surface from a stiff plate to an elastic membrane (Figure 7.4, Mitasova and Mitas, 1993). For very high tension, the surface resembles a rubber sheet with cusps at the data points (Figure 7.4b, tension=160, default smoothing=0.1). For low tension, the surface behaves like a stiff (hard to bend) plate, creating a very smooth surface (Figure 7.4a, tension=10, default smoothing=0.1). Due to its stiffness, it can overshoot in the areas of sharp gradient change (especially if zero smoothing is used, as in Figure 7.6a); in that case, the program gives a warning and increase in tension or smoothing is suggested.

The role of the tension parameter can be also interpreted as a control of the range over which the given point influences the resulting surface. For the high tension, each point influences only its close neighborhood and the surface goes rapidly to trend between the points. This may create cusps around the data points (Figure 7.4b) or steps along the contours (Figure 6.7c). With very low tension, each point has a long range of influence, so it is suitable for interpolation of areas with relatively flat terrain with data points spaced far appart. On the other hand, it may cause visible segments in large data sets (see Section 7.3.4 for a solution). The default input parameters try to adjust the tension to a suitable value based on the analysis of the data point density; however, to fully optimize the tension a more complex procedure based on cross-validation may be used (see Section 7.3.3).

To explore the impact of tension (Figure 7.4), you can interpolate two different surfaces from the Spearfish random elevation data as follows:

```
s.surf.rst elev.rnd.3k elev=elevrnd3k.t10 ten=10
s.surf.rst elev.rnd.3k elev=elevrnd3k.t160 ten=160
```

You can compare these results with the surface computed with the default tension=40 and smoothing=0.1 in our first example in the Figure 7.3c.

Because the tension parameter is scale dependent, it can have different values in different directions, supporting modeling of anisotropic surfaces and volumes (Figure 7.5, Hofierka et al., 2002a). Two additional parameters, angle and scale, were added to `s.surf.rst` in GRASS 5.3 for computation of surfaces with uniform anisotropy, where the new parameters `theta` and `scalex` represent the direction and ratio (scaling) of the anisotropic features.

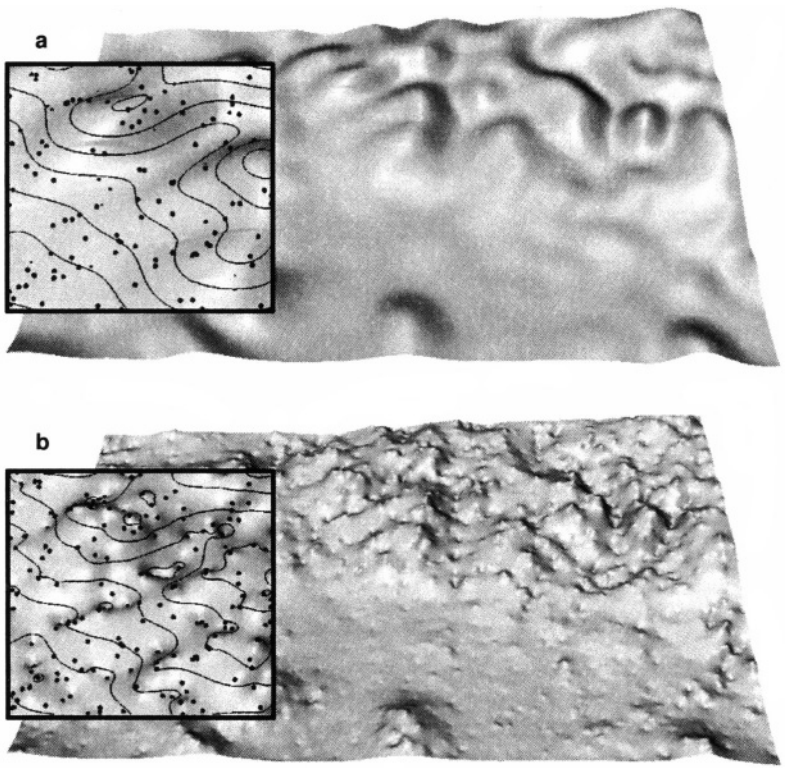


Figure 7.4. Tuning the character of interpolated surface by tension parameter: a) tension=10, smoothing=0.1 leads to a smooth surface with low level of detail useful for modeling major trends; b) tension=160, smoothing=0.1 leads to a surface with cusps (little peaks and pits) in data points, but it is smooth in between (as opposed to IDW). Note that the same smoothing was used in both examples. Compare these results with the surface interpolated with default parameters tension=40, smoothing=0.1 in Figure 7.3c

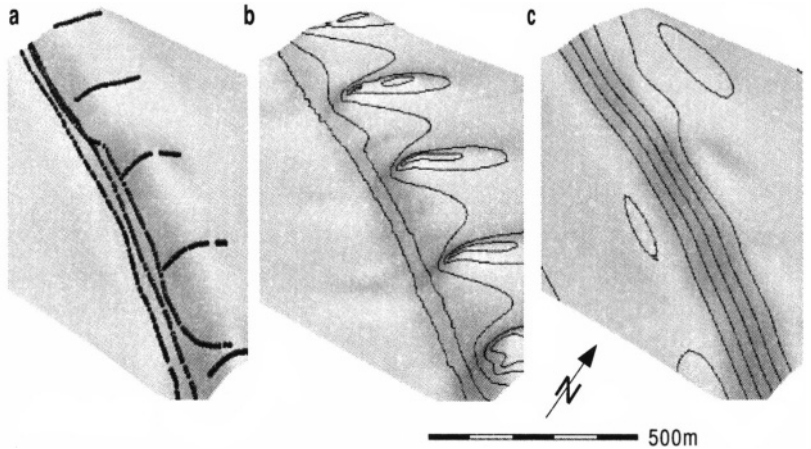


Figure 7.5. RST interpolation of a beach surface surveyed by Real Time Kinematic GPS: a) given data points; b) default parameters in `s.surf.rst`; c) anisotropic tension with angle `theta=160` degrees and scaling `scalex=0.25`

Smoothing parameter. The functionality of smoothing can be illustrated using springs attached to the "pins" representing data points. The higher the smoothing the "softer" the springs and the more is the surface allowed to deviate from the data point in its effort to minimize its energy. GRASS implementation supports the spatially variable smoothing parameter – each site can have different softness of its spring. The interpolation function will pass exactly through the data points that have smoothing set to zero. Uniform smoothing is given as a constant parameter `smooth`, while variable smoothing is given as a floating point attribute in the site file. Smoothing is important when using low tension to prevent overshoots, as well as for removing the noise which may be present in data.

To explore the impact of smoothing, you can interpolate 3 surfaces from the Spearfish random site data using the default tension `ten=40` and the smoothing set to a) `smo=0`, b) `smo=10`, and c) to spatially variable values with `smo=0.1`, if z > 1250 and `smo=10` when z <= 1250 (Figure 7.6, see Figure 7.3c for the result with the default smoothing=0.1):

```
s.surf.rst elev.rnd.3k elev=elevrnd3k.sm0 smo=0
s.surf.rst elev.rnd.3k elev=elevrnd3k.sm10 smo=10
```

You can use `awk` to add the variable smoothing to your site file within the directory `site_lists` in your MAPSET:

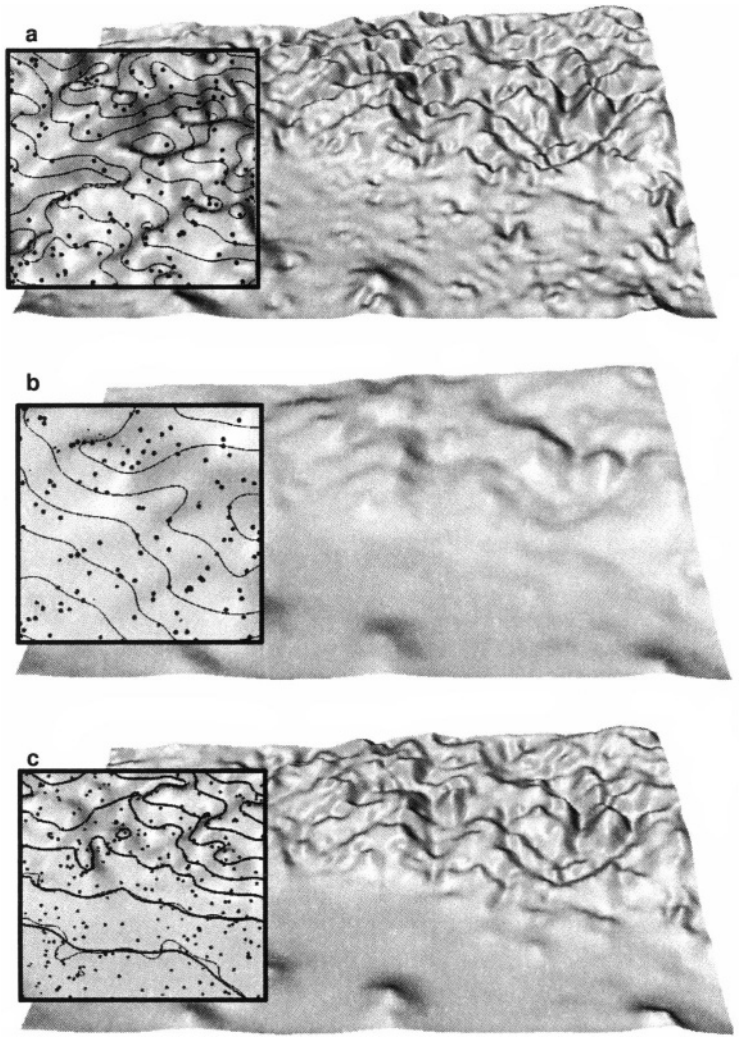


Figure 7.6. Impact of constant and spatially variable smoothing: a) tension=40 and smoothing=0.0: surface passes exactly through the data points, but overshoots may be present; b) tension=40 and smoothing=10.0: surface is very smooth and does not pass exactly through each data point; c) tension=40 and smoothing is 0.1 in the mountainous area and 10.0 in the lowland. Compare these results with the surface interpolated with default parameters tension=40, smoothing=0.1 in Figure 7.3c

```
s.out.ascii -d elev.rnd.3k |\
awk '$3>1250 {sm=0.1} $3<=1250 {sm=10}
{printf "%s|%s|#%s %%%s\n", $1, $2 ,$3, sm}' > elev.rnd.3ksmvar
s.surf.rst elev.rnd.3ksmvar elev=elevrnd3k.smvar smat=2
```

The module `s.out.ascii` pipes the sites to the `awk` program which sets the internal variable `sm` according to the third parameter `$3` as piped from `s.out.ascii`. In our case, this third parameter is the elevation. The next code section within `awk` prints formatted (`printf()` function) GRASS internal sites format. The `%s` are needed to print the values stored in the variables `$1`, `$2` and `$3` as well as `sm` into this formatted string. This procedure is done for every line in the sites file and the output is redirected into a new sites map `elev.rnd.3ksmvar` which has an additional column containing the variable smoothing parameter. In the example with the variable smoothing, surface in the mountains (elevation greater than 1250) is identical with the one obtained from the default settings, on the other hand, the surface in the lowland is much smoother (Figure 7.6c).

7.3.3 Estimating accuracy

Several measures can be used to estimate accuracy of spatial interpolation. The module `s.surf.rst` computes deviations of the resulting surface from the given site data that can be output to a site file `devi` for further analysis. For example, you can compare the deviations of the surfaces generated by the RST default settings (smoothing 0.1) and with a smoothing of 10 by adding the output of the deviation file to interpolation and then computing the summary statistics as follows:

```
g.region rast=elevation.dem -p
s.surf.rst elev.rnd.3k elev=elevrnd3k.def devi=elev.def.devi
s.surf.rst elev.rnd.3k elev=elevrnd3k.sm10 smo=10\
                                devi=elev.sm10.devi
s.univar elev.def.devi
 [...]
      standard deviation 3.13887
 mean of absolute values  2.17833
 [...]

s.univar elev.sm10.devi
 [...]
     standard deviation 21.7113
 mean of absolute values 15.3396
 [...]

r.info elevrnd3k.def
 [...]
   ... rmsdevi=3.138342
 [...]

r.info elevrnd3k.sm10
 [...]
   ... rmsdevi=21.707725
 [...]
```

When you compare the standard deviation and the mean of absolute values of deviations, you can clearly see that the surface with the lower smoothing is closer to the data points than the surface with the high smoothing. Your values of deviations may be slightly different from those published here because your input file, generated by a random procedure, will be slightly different too. Even if the `devi` file is not defined, the standard (root mean square) deviation is written into the *history file* of the resulting raster, along with the minimum and maximum of the values at the given points and in the interpolated raster. Note that the interpolated minimum and maximum values are usually higher or lower than those at the given points, due to the smoothing, especially if the interpolation is performed outside of the area covered by the input data set. The contents of the history file can be retrieved by `r.info`.

The predictive error of the RST interpolation for the given set of parameters can be estimated by a cross-validation procedure (Mitasova et al., 1995) implemented in an experimental version of `s.surf.rst.cv` (available at the NCSU Web site.[2]) The method is based on removing one data point at a time, performing the interpolation for the location of the removed point using the remaining samples, and calculating the residual between the actual value of the removed data point and the estimate for this point obtained from the remaining samples. This procedure is repeated until every sample has been, in turn, removed. The overall performance of the interpolator is then evaluated as the root-mean of squared residuals. Low root-mean-squared error (RMSE) indicates an interpolator that is likely to give more reliable estimates in the areas between the data points. The cross-validation can also be used to find optimal interpolation parameters by minimizing the RMSE (Mitasova et al., 1995, Hofierka et al., 2002a).

7.3.4 Interpolating large data sets (⇑)

Digitized contours or LIDAR (Light detection and ranging) elevation data sets can have over a million data points with the resulting DEMs with thousands of rows and columns. To support processing of such large data sets, `s.surf.rst` and `s.vol.rst` were implemented with a segmented processing procedure. The segmented processing is based on the fact that splines have local behavior, i.e., impact of data points which are far from a given location diminishes rapidly with increasing distance (Powell, 1992). The segmentation uses a decomposition of the studied region into rectangular segments with variable size dependent on the density of data points (Figure 7.7), using *quadtrees* for 2D and *octtrees* for 3D interpolation (Mitasova et al., 1995). For a given segment, the interpolation is carried out using the data points within this segment and from its neighborhood, selected automatically depending on their spatial distribution. Because tension inversely controls the range of influence of data points, this approach requires large neighborhoods to achieve

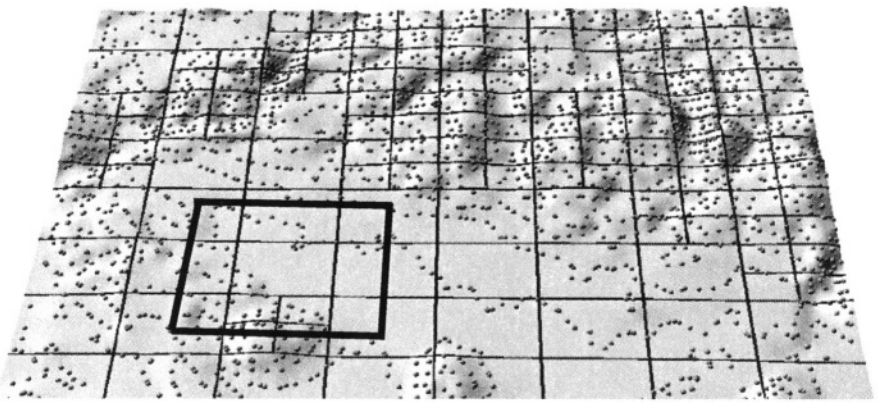


Figure 7.7. Segmented processing of large data sets

smooth connection of segments for very low tension. The number of points in the segment is controlled by `segmax`, the number of points used for interpolation (within the segment and its neighborhood) is controlled by `npmin`. Default values for these parameters usually work, in case that the segments are visible, `npmin` should be increased. Note that while `s.surf.idw` uses 12 given points for computation of a grid value, `s.surf.rst` uses at least `npmin` points with the default `npmin=200`. If the points are dense and homogeneously distributed (as is often the case with LIDAR data), both `segmax` and `npmin` can be set to lower values, leading to substantially faster computation. The IDW computes a new function for each grid point, while RST uses the same function for all grid points within one segment.

Automatized collection of data points, typical for the current digitizing and mapping technologies, leads to substantial oversampling. Therefore, the most effective tool for speeding up the computation is the reduction of data points to the minimum necessary for a given accuracy and resolution. The density of points used for interpolation in `s.surf.rst` is controlled by the parameter `dmin` representing the minimum distance allowed between the data points – points that are closer to each other than this distance are considered identical and not included into interpolation.

As the LIDAR data are becoming increasingly available, we use them to explain the issues related to large data sets and oversampling. Our test data set for a large coastal sand dune called Jockey's Ridge in North Carolina, USA, can be downloaded from the NOAA LIDAR Data Retrieval Tool (LDART) Web site[3], where you can also learn more about the technology. Use LDART to

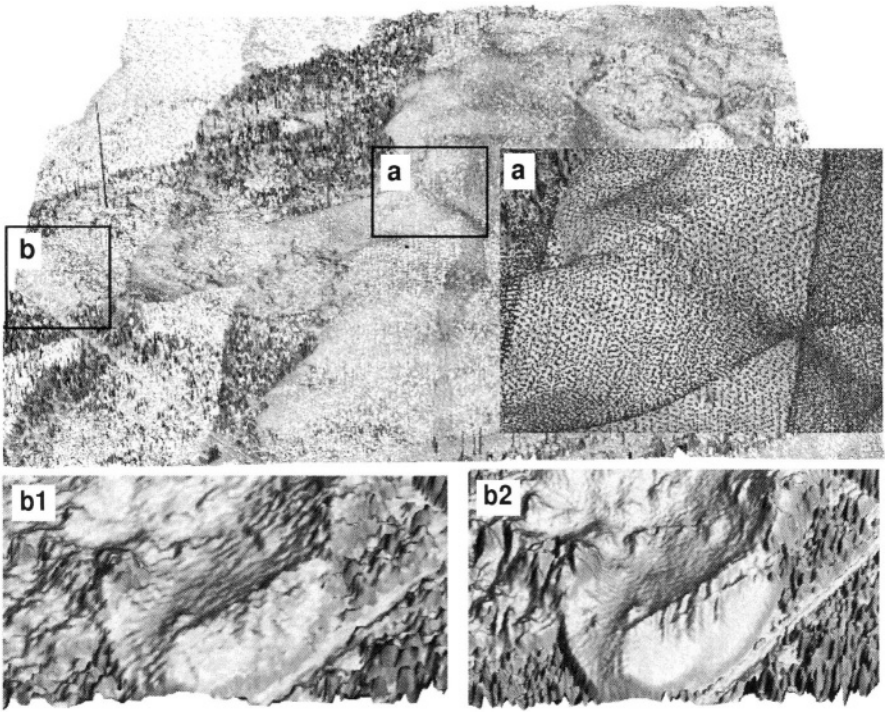


Figure 7.8. Surface created from raw LIDAR data by `s.to.rast` at 1 m resolution with lots of gaps. Insert **a** shows the input point data with overlapping swath in detail. Insert **b1** shows detail of the surface created by `s.to.rast` at 3 m resolution with some loss of detail. Insert **b2** shows detail of a surface computed by `s.surf.rst` at 1 m resolution with high level of detail

select the North Carolina 1999 Fall mission and an area given by the following latitude-longitude coordinates:

```
north:       35.96428
south:       35.95287
west:       -75.63844
east:       -75.62579
```

LDART allows you to select a coordinate system, for example, you can choose UTM zone 18, with the horizontal datum nad83 and vertical datum navd88. Choose bin method `none` to get the point data. To import the data, first create a UTM LOCATION (zone 18, north: 3980167, south: 3978908, west: 442433, east: 443573, resolution 3 m), then use `s.in.ascii` with the `fd=,` option. LIDAR point data create almost continuous coverage of the mapped surface; therefore, you can use `s.to.rast` to get a quick, rough view of the surface

(Figure 7.8). You can do the transformation for 1 m and 3 m resolution as follows (Figure 7.8):

```
g.region sites=lidar99 res=1 -p
s.to.rast lidar99 out=lidar99.sites.1m
d.rast lidar99.sites.1m
g.region res=3 -p
s.to.rast lidar99 out=lidar99.sites.3m
d.rast lidar99.sites.3m
```

With the 1 m resolution, there are lots of gaps in the surface. At the lower resolution, some detail was lost, and there are still a few gaps (Figure 7.8). To interpolate a high resolution DEM, we set the resolution to 1 m and interpolate with RST (Figure 7.9):

```
g.region res=1 -p
s.surf.rst lidar99 elev=lidar99.def &
```

With the default parameters, the interpolation can run for several hours so you should run it in background (<CTRL><Z>, bg). If you drape the given site data over the interpolated surface in `nviz`, you can see that the measured swaths of data overlap and the surface is significantly oversampled (see insert **a** in Figure 7.8). You can reduce the oversampling by increasing the parameter `dmin` from its default value of 0.5 meters (half grid cell size) to 1 or even 2 meters without losing too much detail. The points that are closer to each other than `dmin` are then skipped, leading to faster computation using a smaller number of sites. The most significant speed-up can be achieved by reducing `segmax` and `npmin` values to 20 and 100 respectively, because the high density of LIDAR data does not require big overlaps in segments to achieve the continuity in the resulting surface. If you view the result in `nviz` (see Figure 7.9), you can see the excellent representation of the sharp dune crests and slip faces, even without defining them as breaklines, thanks to the high density of data points. The default values for tension and smoothing preserve the noise produced by laser scanning; to reduce this noise, use higher value of smoothing.

In case of millions of data points exceeding the available RAM, swapping can be avoided by splitting the data into several overlapping strips, each interpolated separately and then patched together (overlap ensures smooth connection). The segmented processing as well as the possibility of splitting the computation into overlapping strips makes the module relatively easy to adapt to parallel processing. The experimental parallel version of `s.surf.rst` developed by Christoph Troyer is available at the NCSU GRASS contributions Web site.[4]

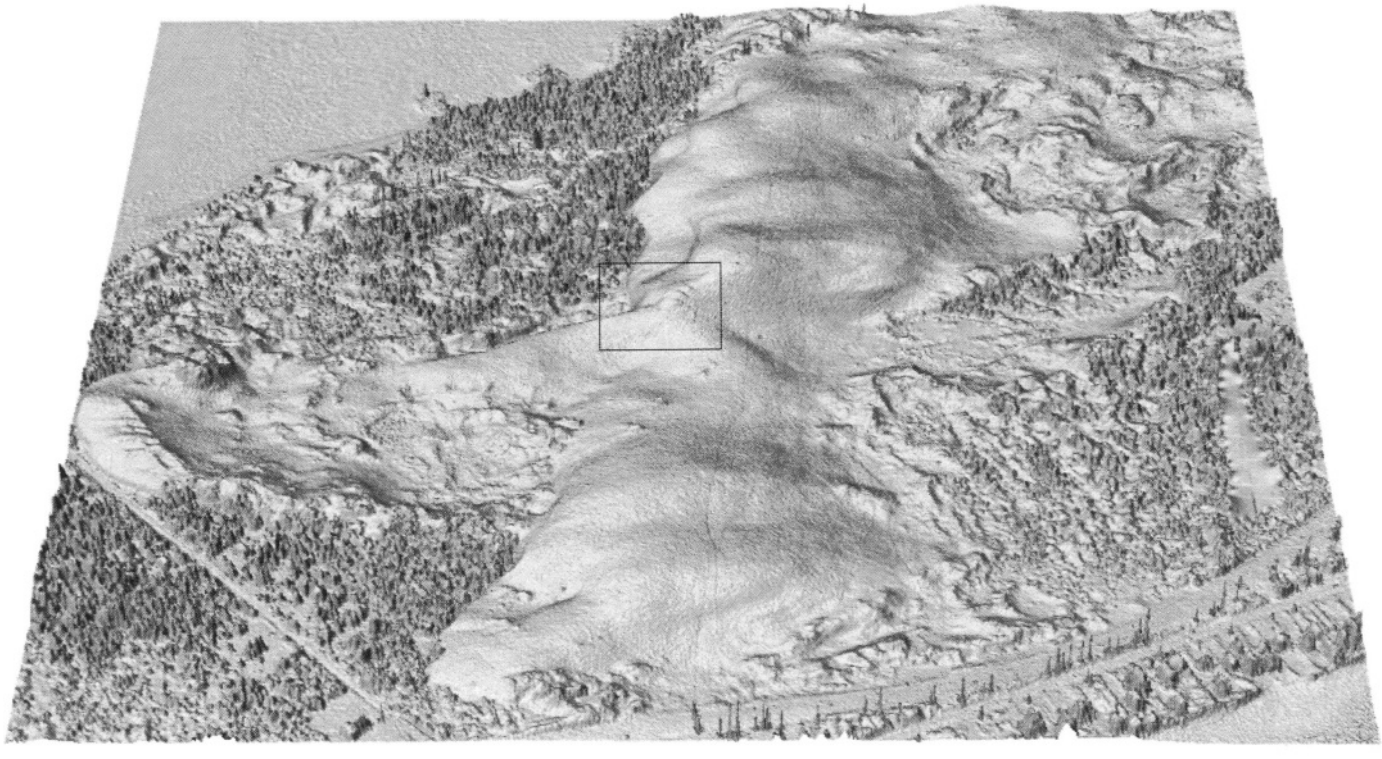


Figure 7.9. LIDAR data interpolated at 1 m resolution using the default settings of `s.surf.rst`. The rectangle is the area used in the previous figure illustrating the oversampling. Data are courtesy NOAA, NASA, and USGS (see the endnote of this chapter)

7.3.5 Surfaces with faults (⇑)

By definition, spline is a smooth function and it has always been regarded as a tool for interpolation of smooth surfaces. However, RST can represent sharp edges or rough surfaces very well, if these features are sufficiently described by the data, as we have demonstrated in the previous section for LIDAR (see for example Figure 7.9).

To create models of surfaces with complex faults pre-defined as vector lines, we can combine `s.surf.rst` with GRASS masking tools. We illustrate the approach using an elevation surface with a deep gully which was surveyed in the field (the data are available at the GRASS Tutorials Web site[5]). First, the points defining the fault need to be extracted from the measured elevation data, based on their attributes.

We can format them as a GRASS ASCII vector line (see Chapter 6) using UNIX text processing tools. Then, we will import this faultline from a file `gully.ascii`, convert it to a raster, and use it as a mask to separate the gully area from the rest of the terrain:

```
v.in.ascii gully.ascii out=gully
v.support -r gully
v.to.rast gully out=gully.mask
```

Use the UNIX text tools to separate the points defining the gully (faultline and gully bottom) and the hillslopes, import them as point files `elev.gully` and `elev.nogully`, and interpolate with RST as follows:

```
s.in.ascii elev.nogully input=nogully.ascii
s.in.ascii elev.gully input=gully.ascii

#interpolate without gully:
s.surf.rst elev.nogully elev=elev.nogully

#interpolate only gully:
s.surf.rst elev.gully elev=elev.gully mask=gully.mask
```

Use a mask for the gully to avoid unnecessary interpolation on the hillslopes where we do not have any gully data. The surfaces are finally merged using map algebra to create a single surface with faults (Figure 7.10):

```
r.null gully.mask setnull=0
r.mapcalc "elev.fault=if(gully.mask,elev.gully,elev.nogully)"
nviz elev.fault
```

7.3.6 Adding third variable: precipitation with elevation (⇑)

Multivariate interpolation is a valuable tool for incorporating the influence of an additional variable. For example, to interpolate precipitation with the influence of topography, the trivariate version of RST `s.vol.rst` can be used.

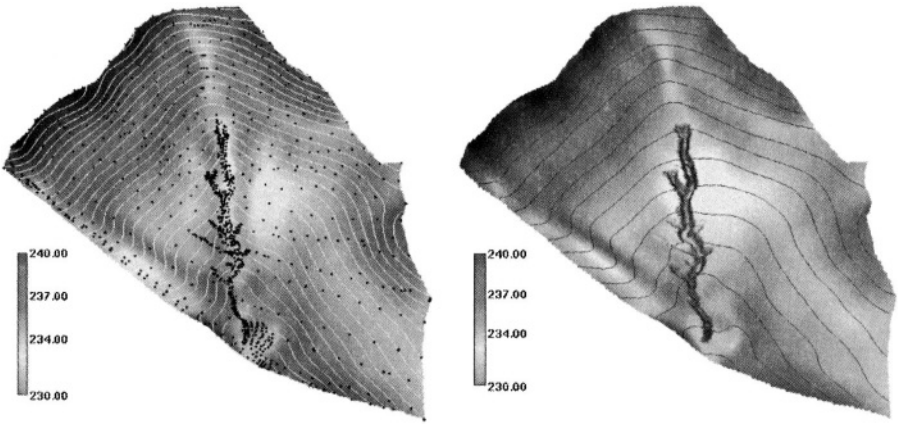


Figure 7.10. Interpolation of a surface with fault representing an edge of a gully. Given field measured data are shown as black dots

The approach is similar to the one proposed by Hutchinson and Bishop, 1983, and it is described in more detail by Mitasova et al., 1995, and Hofierka et al., 2002a.

The approach requires 3D precipitation site data (x, y, z, p) and a raster DEM. The result is a precipitation raster map computed as an intersection of the precipitation volume model with the elevation surface.

As an example we use the computation of long term mean annual precipitation in Slovakia using the data from over 400 meteorological stations and a 500 m resolution DEM, described in detail by Hofierka et al., 2002a (the data are available for download on the GRASS Tutorials Web site, see Endnotes). The input site data `precip3d` include 3D coordinates for each meteorological station and mean annual precipitation as a floating point attribute, while the `dem500` DEM is provided as a 500m resolution raster (Figure 7.11a). The output is a 2D raster map representing precipitation called `precip.topo` (Figure 7.11c). The 2D region should be set to the resolution of the input DEM. The module also expects that a 3D region is defined. To create it within your current MAPSET, you have to run the `g3.createwind` script. To ensure that the interpolation is performed only for the area of the Slovak republic, define the mask using the provided raster file `mask`. The resulting 2D precipitation raster map can then be computed by `s.vol.rst`:

```
g.region rast=dem500 -p
g3.createwind t=3000 b=0 dres=3000
g.copy rast=mask,MASK
```

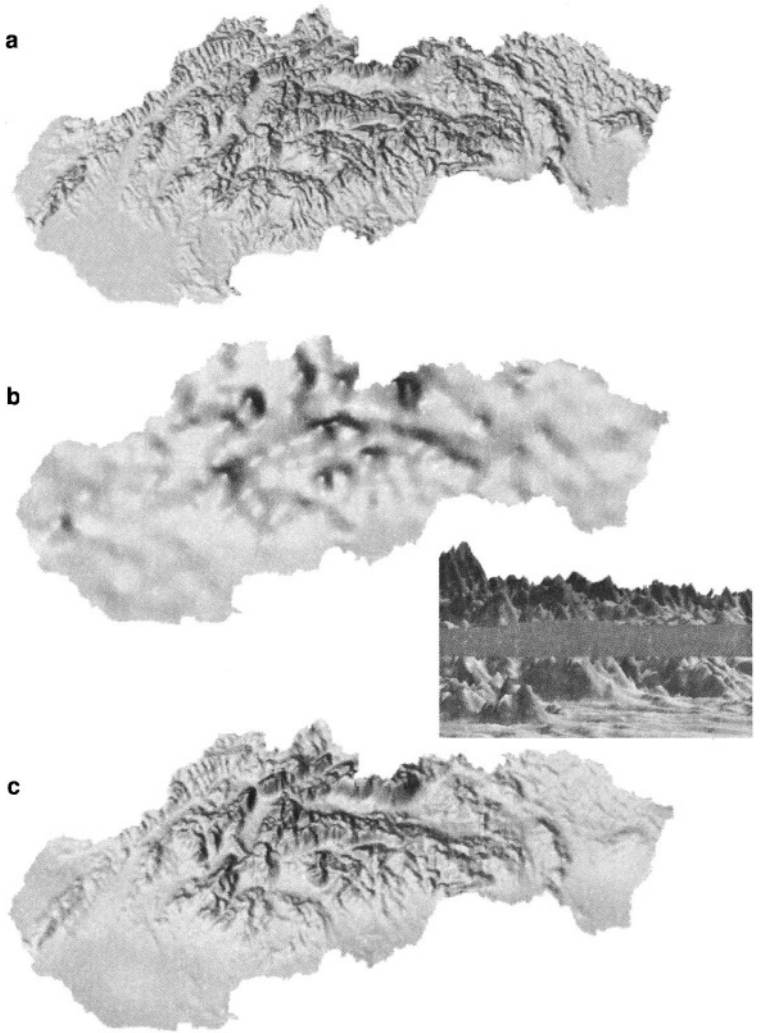


Figure 7.11. Interpolation of precipitation with influence of topography: a) Input 500 m resolution DEM; b) precipitation distribution using bivariate interpolation by `s.surf.rst`; c) precipitation with influence of topography interpolated by `s.vol.rst`. Precipitation is displayed as a surface ("hills" are high precipitation values). The insert shows elevation surface intersected by 700 mm precipitation isosurface

```
s.vol.rst in=precip3d cellinp=dem500 cellout=precip.topo \
          zmult=50 segmax=700
```

The horizontal resolution is defined as 500 m with `g.region` and the vertical resultion `dres` as 3000 m with `g3.createwind`, creating a volume with a single layer, starting at 0 m elevation and ending at 3000 m. The segmentation can be skipped by using `segmax=700`, because the number of input data points is only 435. The impact of topography on the resulting precipitation map is controlled by the vertical scaling parameter `zmult`, as well as by the resolution and smoothing of the DEM as demonstrated by Hofierka et al., 2002a. The resulting map `precip.topo` is a 2D raster map representing the precipitation. The 3D RST module internally computes a 3D (volume) precipitation function. When using the `cellinp` option, the precipitation values are extracted from the precipitation volume at the elevations given by the `dem500` raster map (see Figure 7.11). You can experiment with tension, smoothing, and zmult parameters to get the best result. This approach captures a more complex, spatially variable relation between precipitation and elevation than the traditional methods that are based on statistical correlation.

7.3.7 Volume and volume-temporal interpolation (⇑)

GRASS provides some limited experimental tools for working with volume data. We describe the functioning tools here to encourage further development. Some of the prototype applications are demonstrated at the Spatial interpolation Web site.[6] To illustrate the volume data processing tools we use the Chesapeake Bay nitrogen concentration data that can be downloaded from the related GRASS Tutorials Web site (see Endnotes), see also an example at the Multidimensional Spatial interpolation Web site. First, you need to set up a LOCATION based on the metadata and import the site file. Then define the 3D region using `g3.createwind` and `g3.setregion`, and you can convert the 3D point data (x,y,z,w) to discrete voxel representation with `s.to.rast3`:

```
g3.createwind b=-33 t=1 dres=2
g3.setregion b=-33 t=1 res=10 dres=2
s.to.rast3 nitro3d out=nitro3d.vol
g3.list
```

The command `g3.list` allows you to list the volume raster files that are available. The continuous volume from 3D scattered point data can be created by interpolation to 3D raster using the IDW method implemented as `s.vol.idw`, or the RST method using `s.vol.rst`, with RST usually providing more accurate results. The mathematical description of the method, including the equations for computation of associated gradients and curvatures, is in the Appendix B. Trivariate RST has similar properties and parameters as the bivariate version, so the principles described in the previous sections apply here as well (see for example impact of tension in 2D and 3D in Mitas and Mitasova, 1999).

To illustrate the volume interpolation, you can create a volume model of spatial distribution of the Chesapeake Bay nitrogen concentrations.

To limit the interpolation to the water body, import the shoreline data `chesapeakeshore.ascii` using `v.in.ascii` and create a mask for the Bay using `v.to.rast`. Then you can interpolate the masked volume using `s.vol.rst`:

```
v.in.ascii chesapeakeshore.ascii out=shore
v.support -r shore
v.to.rast shore out=shore.mask
s.vol.rst nitro3d elev=nitro3d segmax=400 zmult=1000\
                                    maskmap=shore.mask
```

Note that the site data should be in the correct 3D format: `x|y|z|#n %w1` and that the vertical resolution of the resulting grid is much higher than the horizontal resolution. Parameter `zmult` is set so that the vertical distances between the data points are of the same magnitude as the horizontal distances to ensure the stability of interpolation. See the Section 8.2.4 for various possibilities to visualize the results.

Similarly to the bivariate version, trivariate RST can compute a number of parameters related to the gradients and curvatures of the volume model. An experimental version of RST interpolation for 4D data (volumes changing in time) `s.volt.rst` is also available for update and further development.

7.3.8 Geostatistics and splines

As we have described in the previous sections, GRASS provides fully integrated spline interpolation and a wide range of geostatistical tools, including kriging through the link with Open Source geostatistical software (see Chapter 13). Because the relation between splines and kriging is a frequently asked question, we provide here a brief explanation.

Several authors (e.g., Wahba, 1990; Cressie, 1993) have demonstrated that splines are formally equivalent to universal kriging with the choice of the covariance function determined by the smoothness seminorm (also called roughness penalty). Therefore, many of the geostatistical concepts can be exploited within the spline framework.

Kriging assumes that the spatial distribution of a geographic phenomenon can be modeled by a realization of a random function and uses statistical techniques to analyze the data (drift, covariance) and statistical criteria (unbiasedness and minimum variance) for predictions. However, subjective decisions are necessary (Journel, 1996) such as judgement about stationarity, and choice of a function for theoretical variogram (variogram model). Kriging is therefore successful for phenomena with a strong random component and/or for the problems where estimation of statistical characteristics (uncertainty) is the key.

Splines rely on a physical model with flexibility provided by change of elastic properties of the interpolation function. Often, physical phenomena result from processes which minimize energy, with a typical example of terrain with

its balance between gravitation force, soil cohesion, and impact of climate. For these cases, splines proved to be rather successful. Moreover, splines provide enough flexibility for local geometry analysis, which is often used as input to various process-based models.

However, most of the surfaces or volumes are neither stochastic nor elastic media, but they are results of a host of natural (e.g., fluxes, diffusion) and/or socioeconomic processes. Therefore, each of the mentioned methods has a limited realm of applicability and, depending on the knowledge and experience of the user, proper choice of the method and its parameters can significantly affect the final results as illustrated by numerous examples throughout this book.

NOTES

1 Multidimensional Spatial Interpolation
`http://skagit.meas.ncsu.edu/~helena/gmslab/viz/sinter.html`

2 J. Hofierka, 2003, Crossvalidation for s.surf.rst: s.surf.rst.cv.tar.gz,
`http://skagit.meas.ncsu.edu/~helena/grasswork/grasscontrib/`

3 LIDAR data,
`http://www.csc.noaa.gov/crs/tcm/about_ldart.html`
Airborne Topographic Mapper LIDAR data were collected in partnership with the National Oceanic and Atmospheric Administration (NOAA) Coastal Services Center, the NASA Wallops Flight Facility, the U.S. Geological Survey (USGS) Center for Coastal and Regional Marine Geology, and the NOAA Aircraft Operations Center

4 Ch. Troyer, 2003, Parallel s.surf.rst: rstmods2fixed.tar.gz,
`http://skagit.meas.ncsu.edu/~helena/grasswork/grasscontrib/`

5 GRASS Tutorials Web site,
`http://mpa.itc.it/grasstutor/`

6 Spatial interpolation:
Chesapeake Bay Nitrogen,
`http://skagit.meas.ncsu.edu/~helena/gmslab/viz/ches.html`
Concentrations of chemicals,
`http://skagit.meas.ncsu.edu/~helena/gmslab/viz/vol1.html`
Soil properties,
`http://skagit.meas.ncsu.edu/~helena/gmslab/gsoils/ccsoil2.html`

Chapter 8

GRAPHICAL OUTPUT AND VISUALIZATION

Visual analysis and communication based on graphical output is a core component of GIS. Graphical representation of georeferenced data and creation of cartographic models provides important means for understanding and communicating complex spatial relationships. GRASS includes a wide range of graphical tools from simple two-dimensional display to sophisticated visualization and animation.

8.1. TWO-DIMENSIONAL DISPLAY AND ANIMATION

The most common approach to viewing and visually exploring geospatial data is an interactive display of two-dimensional images using color and different area, line and point symbols. This approach is mostly based on traditional cartography, with current computer graphics tools offering greater flexibility in color, symbols, dynamics and interactivity that was not possible with the traditional maps.

8.1.1 Displaying map layers using the GRASS monitor

In the previous chapters, we have already described the basic tools for viewing map layers, such as `d.rast`, `d.vect`, and `d.sites`. In this section, we just add few notes and focus on additional tools.

To simultaneously view more than a single raster map layer (for example to compare patterns), it is possible to open up to seven GRASS monitors named `x0` through `x6` using the command `d.mon`. Use the parameter `select` to choose in which monitor the map layer will be displayed, for example (Spearfish LOCATION):

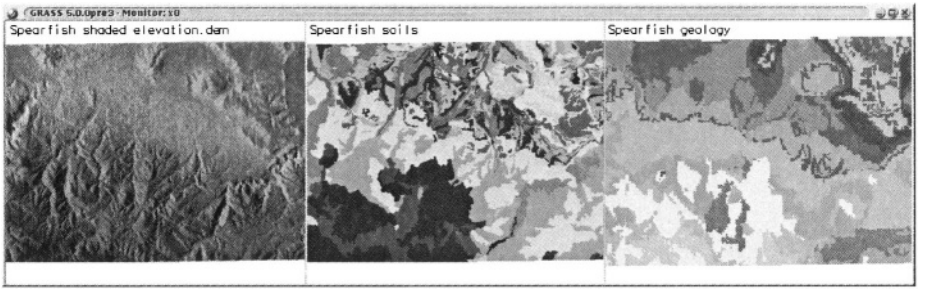


Figure 8.1. Map display with `d.frame`: three frames with shaded DEM, soils and geology map (Spearfish data set). The titles are written with `d.text`

```
d.mon x0
d.rast soils
d.mon x1
d.rast geology
d.mon select=x0
d.vect roads
```

will display soils map in monitor `x0`, the geology map in monitor `x1`, and overlay the roads map in monitor `x0`. If you are planning to use multiple monitors regularly, it is worth trying the module `d.dm` which provides a graphical user interface for managing multiple monitors and browse through multiple map layers interactively.

Each monitor can be split into several frames using the `d.frame` command. Using the flag `-c`, you can subdivide the GRASS monitor into rectangular areas (frames) by mouse. Subsequent display commands will be applied only to the latest defined or selected frame. In case that you have several frames, select another one with `d.frame -s` and click with left mouse button into the desired frame, accept it with right mouse button. To remove all frames, run `d.frame -e`. As an example, to split the current monitor into two frames, you can define the following:

```
d.frame -e
d.frame -c at="0,100,0,50"
d.frame -c at="0,100,50,100"
d.frame -s

Buttons:
Left:   Identify a frame
Middle: Keep original frame
Right:  Accept frame
```

The frame coordinates are defined in percent of the monitor size (bottom, top, left, right). Using this approach you can define frame regions independent

from the true monitor pixel size. Before displaying maps in a frame, select it with the mouse as described above. An example with three frames in a GRASS monitor is shown in Figure 8.1.

As with any other window, you can adjust the GRASS monitor size using the mouse. To change its default size, use the UNIX environment variables `GRASS_HEIGHT` and `GRASS_WIDTH` either in `/etc/profile` or locally in `$HOME/.grassrc5`. If you are using `tcltkgrass` you can set the size in the CONFIG ⇝ OPTIONS ⇝ DISPLAY DIMENSIONS menu. These changes will become valid for your next start of a GRASS monitor. In order to save this new configuration for future sessions, you have to answer YES to SAVE CONFIG when leaving `tcltkgrass`.

You can add legend, scale, and text to your displayed map using `d.legend`, `d.barscale`, and `d.text` respectively. If you want to include this map in a presentation or a report, you can use a graphical program such as `gimp` to snapshot the window. Later, in Section 8.1.3, we show how to generate high resolution output as an image file with the PNG driver. You can display the legend in a separate monitor, frame, or you can just stretch your current monitor to make space for the legend and place it with a mouse using the `-m` option. For `d.legend`, you can use continuous colors for the continuous field data with the flag `-s` and discrete colors for raster map layers with categories (see examples in Chapter 12). For floating point raster maps representing continuous fields, it is appropriate to use the legend command with the `-s` option, creating a legend with smoothly changing colors, because in such maps the colors for each cell value are interpolated based on the values and colors given in the color table.

There is no legend tool yet for vector and site data – you need to use external graphical software to add it to the snapshot image. You can display labels for your raster, vector, or sites map layer by using `d.rast.labels`, `d.vect.labels`, and `d.site.labels`.

For raster data, working with color provides a powerful tool for extracting important spatial information and communicating it effectively. As we have explained in Section 5.1.1, the raster color table can be defined by `r.colors`. Besides selection from a set of predefined color tables you can define the colors by their names using the option `color=rules`, or you can copy a color table from another raster map using the option `raster=mymap` (see examples in Section 5.1.1). For a refined definition of colors you can use the red, green, blue (RGB) color description (see Section 9.7.1). To find suitable RGB values for a desired color, use any graphics tool provided by your system. For example, in `gimp`, find a palette under "File" ⇝ "Dialogs" ⇝ "Palette". Select a palette suitable for your map and then click on the individual colors to get the RGB values which you can then use in `r.colors`. Besides the applications throughout this book, you can find examples of rules for creating the color ta-

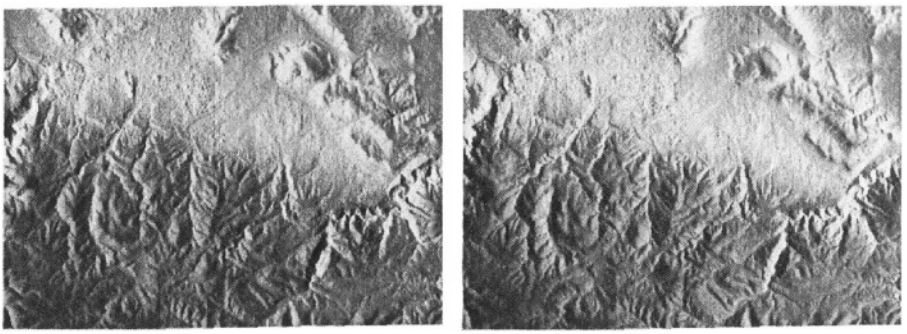


Figure 8.2. Shaded elevation maps: shade map with sun azimuth=270° from north (left) and shade map with sun azimuth=90° (right), sun altitude=30° above horizon (Spearfish data set)

bles in the Chapter 12, in the Section 5.4.4 and, of course, in the manual page for `r.colors`.

8.1.2 Creating a 2D shaded elevation map

To enhance the perception of topography represented by a DEM, a shaded elevation map can be generated quite easily. A special color transformation is used to prepare a translucent view of the DEM (or any other raster map) and the shade map. It is based on the IHS color transformation which is explained in greater detail in Section 9.7.1. First, we generate a shade map based on the sun position using the script `shade.rel.sh`. The name of the resulting shade map is created automatically by adding `.shade` name extension to the name of the elevation file. This map is then used to display the `elevation.dem` map layer with shaded topography by `d.his`:

```
shade.rel.sh altitude=30 azimuth=270 elevation=elevation.dem
d.rast elevation.dem.shade
d.his h=elevation.dem i=elevation.dem.shade
```

You may experiment with different values for `altitude` and `azimuth` when creating the shade map to highlight various topographic features. Figure 8.2 shows the effects for different sun azimuth angles. You can also apply shading to other types of surfaces when studying their structure.

If you want to save the shaded map into a file, use `r.his` instead of `d.his`. It creates three map layers representing the red, green and blue channels (because the original map is 24bit, it writes three 8bit maps). In our next example, we call them `el.b`, `el.g`, `el.r`. You can then use the module `r.composite` to combine the three color maps within GRASS into a single shaded elevation map `dem.shaded`:

```
r.his h=elevation.dem i=elevation.dem.shade b=el.b g=el.g r=el.r
r.composite b=el.b g=el.g r=el.r out=dem.shaded
d.rast dem.shaded
```

The module `r.composite` provides optional parameters to control the color levels to be used for each color component (default color levels per channel: 32). This default number of levels results into a total of 32768 possible colors (equivalent to 15 bit per pixel). Due to limitations in the GRASS display color model both `r.composite` and `d.rast` will significantly slow down if more colors are used. However, for human eye, this number of grey shades is quite sufficient. You can also export the three map layers and compose them into 24 bit shaded elevation image using external graphics tools. Alternatively you can export the map using `r.out.ppm3` which writes a 24 bit PPM file. Note that the composite shaded elevation map is only usable for visualization purposes as the "elevation" cell values are modified due to the shading.

8.1.3 Monitor output to PNG and HTML files (⇑)

Besides the GRASS monitor, it is possible to output the map display to other types of graphics drivers such as PNG or HTMLMAP (read more about the drivers in the GRASS 5.3 User manual[1], or by running `g.manual drivers`).

PNG file driver. While the regular GRASS monitor displays the raster map layer at the resolution given by your display system, the PNG driver was implemented to create a user defined, high resolution output in PNG image format. It uses the PNG library[2]. True color output is supported. The PNG format (Portable Network Graphics) is a lossless, highly compressing image format designed as a replacement for GIF and TIFF which may contain patented algorithms (note that the JPEG algorithm compression is lossy and is often inadequate).

The use of the driver is similar to the use of the GRASS monitor with the output stored in a PNG file when the PNG driver is stopped. For example, you can create a PNG image with elevation and soil map layers as illustrated by the following sequence of commands. First, start up the driver (here syntax for `bash` shell):

```
export GRASS_TRUECOLOR="TRUE"
d.mon start=PNG
d.mon select=PNG
```

Then, display a raster map and a vector map from the Spearfish data set:

```
d.rast elevation.dem
d.vect soils color=blue
```

The PNG file called map.png will be automatically written into your current directory when you stop the driver:

```
d.mon stop=PNG
```

The resolution of the resulting image, its background color, true color support, and the name of the output file are controlled by GRASS variables and UNIX environment variables (see the manual page for the PNG driver). For example, to output your raster map as an image called `myimage.png` with the size of 3000 x 4000 pixels, you need to set your environmental variables as follows:

```
export GRASS_WIDTH=3000
export GRASS_HEIGHT=4000
export GRASS_PNGFILE=myimage.png
```

Otherwise your output will use the default settings; that means 8 bit colors, the 640 x 480 image size, and the file name map.png.

HTMLMAP driver. The HTMLMAP driver supports the generation of HTML image maps for area vector data. In these maps, different portions of an image are linked to URL targets, such as web sites or web documents.

The driver can be used only with the following GRASS display commands: `d.text` to pass HREF information for resulting image maps and `d.area` to draw polygons from a vector map. The use of this driver is similar to the use of the GRASS monitor, but it stores all output to a HTMLMAP file, as illustrated by the following example. First, start up the driver:

```
d.mon start=HTMLMAP
d.mon select=HTMLMAP
```

Then, you can display two vector maps along with strings to define the target URLs:

```
echo "http://www.dummyurl.dum/gis/" | d.text
d.area map=trn.sites
echo "http://www.anotherdummy.dum/lab/map.html" | d.text
d.area map=rstrct.areas
```

Pipe the text defining the target URLs using the UNIX `echo` command; alternatively, you can just type `d.text` and provide the parameters defining the size and color of the text, as well as the text itself ending it with EOF (<CTRL><D>). To write the HTMLMAP file to your current directory, you just stop the driver:

```
d.mon stop=HTMLMAP
```

The size of the image, output file name, and the HTML type (client, apache, raw) are defined by GRASS environment variables, as they are for the standard GRASS monitor or PNG driver (see HTMLMAP driver manual entry). By default, the map will be written to a file called `htmlmap` as a client type of image.

The image map can be combined with an underlying raster image (`backmap.png` in our example) created by the PNG driver and inserted into a HTML document with reference to the HTML map. The basic structure is:

```
<HTML>
<BODY>
<img src="backmap.png" usemap="#map" alt="grass">
<MAP NAME="map">
<AREA SHAPE="POLY"
 HREF="http://www.dummyurl.dum/gis/"
[...]
</MAP>
</BODY>
</HTML>
```

The web user will see the `backmap.png` in the browser, links are related to the defined area map. Obviously both should be related to each other.

8.1.4 Animations in 2D space

If you have a series of data (temporal, spatial, parameter scans) you can animate them using `xganim`. The command loads a specified series of GRASS raster map layers and then animates the series. Up to four different map series can be animated simultaneously – a task often needed when analyzing outputs from simulations of landscape processes. For example, you can animate the evolution of water flow and discharge, stored in raster files `hh00100` through `hh01000`, and `qw00100` through `qw01000` in the Spearfish data set, as follows:

```
xganim view1="hh0*" view2="qw0*"
```

You can either list all the map layers by name or use wildcards, however, you need to be careful about the numbering system that you use to ensure proper order of map layers, and keep the possibility to insert additional layers. The program provides a simple interface with controls for speed, looping, direction of play, running and stopping the animation, and stepping through the frames. If the animation is "jumpy", it is usually because your pattern does not change smoothly – you can add additional frames by interpolating between the map layers. Often, a simple average between two raster maps is sufficient (use `r.mapcalc`). Animation is also useful for browsing through a larger set of map layers and as a preview tool when preparing data for animation in 3D using `nviz`. To save your animation slides as an MPEG file, use `r.out.mpeg` (see the manual how to use wildcards).

8.2. VISUALIZATION IN 3D SPACE WITH NVIZ

The advanced, interactive visualization tool `nviz` can be used to view the data in 3D space and to perform visual analysis of multiple surfaces, vector and site map layers. The module is fully integrated with the GRASS data structure and runs directly from the GRASS prompt. It also supports scripting for producing dynamic visualizations via animation. To learn how to use `nviz` read *Nviz tutorial*[3] that can be accessed by clicking on the "help" button of its interface. To get you started, we provide a brief overview of `nviz` capabilities with few examples. You can find numerous images created by `nviz` throughout this book. The module provides the following capabilities:

- visualization of 2D raster map layers as multiple surfaces in 3D space, with the capability to use different data sets for surface topography, surface color and transparency;
- interactive positioning, zooming, and z-scaling;
- interactive lighting with adjustable light position, color, intensity and surface reflectivity;
- display of multiple vector map layers draped over selected surfaces or flat at a selected height;
- display of multiple site data layers draped over selected surfaces or in viewing 3D in their 3D location;
- animation capabilities with two options:
  - key-frame animation for creating fly-by's,
  - scripting for automatically generating complex animations from series of map layers;
- interactive query of raster data displayed as a surface and color;
- interactive slicing through multiple surfaces using cutting planes.

The `nviz` module is enhanced quite frequently, so some differences between the latest version and our description are possible. The most important updates will be posted on the GRASS Tutorials Web site.[4]

8.2.1 Viewing multiple map layers

You can start the program without defining any map layers by

```
nviz -q
```

and use the interface to load your data. Alternatively, you can define the map layers that you want to visualize on the command line; for example, to view the DEM with streams, roads, and some site data from our Spearfish data set, run:

```
nviz elevation.dem vect=streams,roads sites=archsites,bugsites
```

The program opens a graphics window with a coarse model of the elevation surface and a control panel window. If you click on the "DRAW" button, the elevation model with the vector and site data draped over it will be shown at higher resolution. Depending on the size of your raster map, the surface may not be rendered at its full resolution and the view may not be optimal. In the following paragraphs, we explain how to adjust it to create a desired 3D view of studied area.

Controlling the view. The position of the viewing point, viewing direction, perspective (zoom-in, zoom-out) and tilt of the surface can be adjusted using the "Controls" menu (Figure 8.3). Use the left mouse button to move the puck around the viewing direction square to change the position of viewer and the direction of view – the coarse model of your DEM will move simultaneously making it easier to find the desired viewing position. The "perspective" slider allows you to zoom-in and zoom-out while the "height" slider controls the viewing height. The "zexag" slider is used to interactively modify the z-exaggeration (it effectively multiplies the elevation data); note that it also changes the height of the view so you may need to re-adjust it. If you cannot achieve the desired view with the sliders or if you want to define an exact value for any of the viewing variables, you can type them into the related field and type <ENTER> to continue. To focus on an area off the center of your map, use the "look here" button to select a new center of view with a mouse click (pin your surface in your focus area). All movement of the surface will be centered around this point. To get an ortho view, click on the "top" button and use "RESET" button to get back to the default behavior. The "twist" slider allows you to tilt the displayed surface to simulate the view from a turning airplane.

The surface with all other map layers will be rendered after each change. If you are using multiple steps to adjust the view of your surface, you can switch off this automatic rendering using the buttons in the upper part of the control menu.

Modifying properties of surfaces. The viewed surfaces are managed using the options provided by the "Panel" ⇝ "Surfaces" menu (Figure 8.4). In the upper part of this menu you can adjust the drawing style as well as the level of detail for the rendered surface. By default, the surface is rendered as colored polygons with Gouraud (smoothed) shading, using a coarser resolution while the surface is interactively manipulated. To change the surface display to a

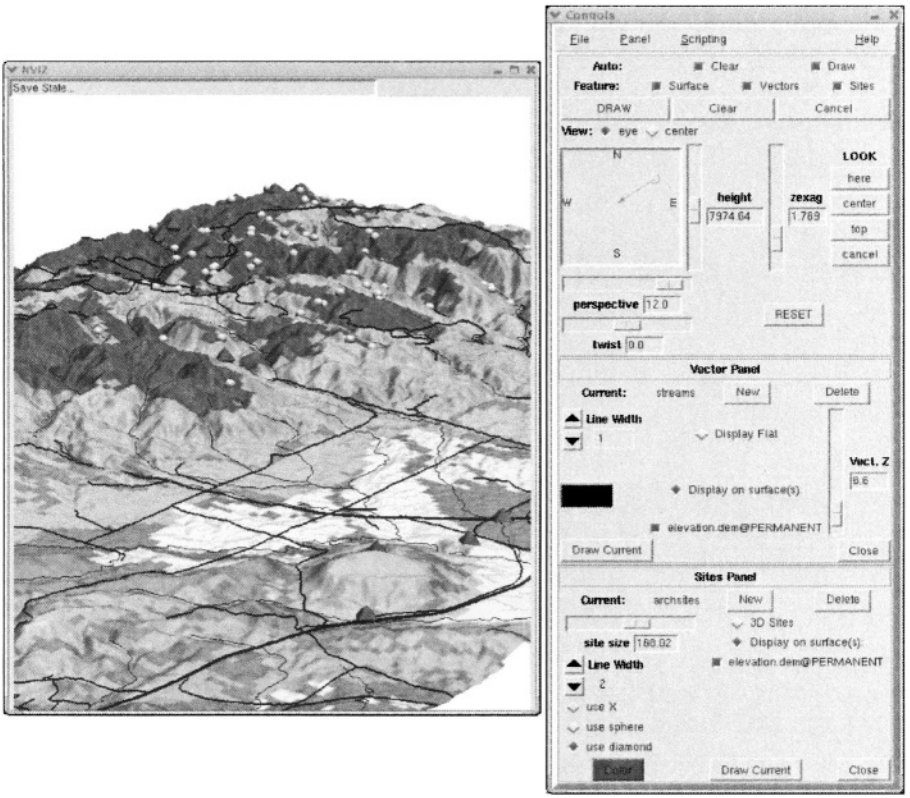


Figure 8.3. Spearfish geology map draped over a DEM with overlayed streams and roads as vector data, and archaeological and insect collection sites as point symbols (pyramids and spheres respectively)

mesh (wire) or colored surface with a mesh, select the desired option from the menu under "Surface style". For fast interactive manipulation, you can select "wire" from the "Grid style" menu. To render the surface at the current region resolution (as given by `g.region`), set the "Polygon Resolution" to "1". Note that if the current resolution is higher than the resolution of the raster file used for topography, the raster file is automatically resampled leading to the discontinuous surface shown by Figure 5.3 (see Chapter 5). To speed up the rendering while exploring the viewing parameters you can lower the rendering resolution (increase the cell size) by choosing a higher value of polygon resolution. If you are using any style that involves wire, you can adjust its grid spacing with "Grid Resolution".

To drape a new color map over the surface select a new raster map using the "color" option from the "Surface Attribute" menu, for example `soils.ph` in

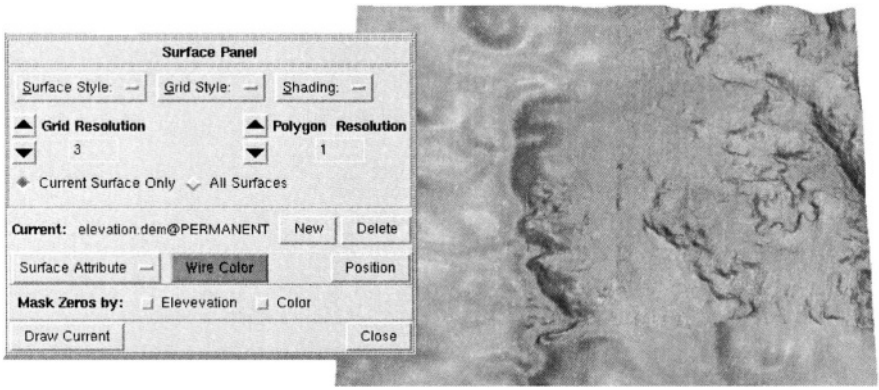


Figure 8.4. Displaying topography at multiple resolutions controlled in the upper part of the *Surface* menu, using multiple, masked-out surfaces

our Spearfish example. After loading the raster map, use "DRAW" to render your elevation surface with the new color map (Figure 8.3). To overlay an additional raster map, you can use transparency. However, for its meaningful application, the raster map for transparency should be fairly simple. A possible use may be to suppress (lighten) the areas outside a studied watershed.

To render only a subset of the surface, you can define a raster map to be applied as a mask using the "mask" option from the "Surface Attribute" menu. Masking can also be used to create a 3D view of topography with spatially variable resolution using nested grids approach (Figure 8.4). For example, assume that you have a 100 m resolution DEM for entire region and a 20 m resolution DEM for a smaller subarea. To create a multiresolution surface, set the resolution to 20 m by `g.region` and start `nviz -q`. Switch off all automated redraw buttons in the top panel and load the two DEMs using the "Panel" ⇝ "Surface" ⇝ "New" button. Then, with the current surface set to the 100 m resolution DEM, define its mask using the 20 m DEM through the "mask" option from the "Surface Attribute" menu. Set "Polygon Resolution" to 5 (5 x 20 = 100) and render the 100 m resolution DEM using the "Draw current" button. You will get the surface with a "hole" in the area where the high resolution DEM is located. Then, select the 20 m DEM as current, set "Polygon Resolution" to 1 and render the 20 m resolution DEM with the "Draw current" button and save the image if it looks good. Depending on resolution, there may be a masked strip between the two surfaces, you can use a slightly smaller raster than the inserted DEM as a mask to ensure sufficient overlap.

Displaying vector data. If you would like to choose a different color for your vector map or add an additional vector map layer, open the vector panel by choosing "Panel" ⇝ "Vectors" (Figure 8.3). You can load the new vector map using the "new" button. After loading, the switch "Display on surface(s)" will be automatically activated for all surfaces (when working with multiple surfaces switch off those on which you do not want to drape your new vector data). You can adjust the line width and color by using the appropriate buttons. For example, select black for `roads` and then click on the name of the current vector file, select `streams` from the menu of available layers and change its color to blue. You can use "Draw current" in the vector panel to drape only the selected vector file. To render the surface with all vectors and sites draped over it, use the "Draw" button on the top of the movement panel. If the lines are not fully rendered, move them slightly above the surface using the "Vect.Z" slider.

Displaying sites. To change the symbol used for the site data, go to "Panel" ⇝ "Sites" and open the site panel (Figure 8.3). It is similar to the vector panel – you can use it to add or delete a sites map layer, select the surface on which the current site data should be draped, and select the symbol, including its color and size. If you have 3D site data, you can also display them in the 3D position rather than draped over the surface. This is particularly useful when evaluating interpolation and smoothing of surfaces as you can see how much the surface deviates from the given data and where the highest deviations are.

Controlling light. Interactive light manipulation is useful for detecting noise or small errors in surfaces as well as for enhancing the 3D perception of surface topography and creating special effects. The panel "Panel" ⇝ "Lights" is used to change the lighting parameters such as the light color, brightness, ambient (dispersed) light and a position of light source (height and direction) by using the appropriate sliders and a square with puck. Interactive adjustment of lighting is made easier by a sphere which appears in the center of the graphics window and shows the current lighting effects as the changes are made (Figure 8.5). The sphere disappears when the surface is rendered, for example with the "Draw" button.

Adding legend and labels. To add a legend, title, or other text to the images displayed by `nviz`, open the "Panel" ⇝ "Label" panel that allows you to type in short text and place it in the graphical window with the mouse. The legend has options similar to the command `d.legend` and you can again place it with the mouse. The legend is then automatically redrawn with the image, to remove the legend, draw the surface with the button "On" switched off.

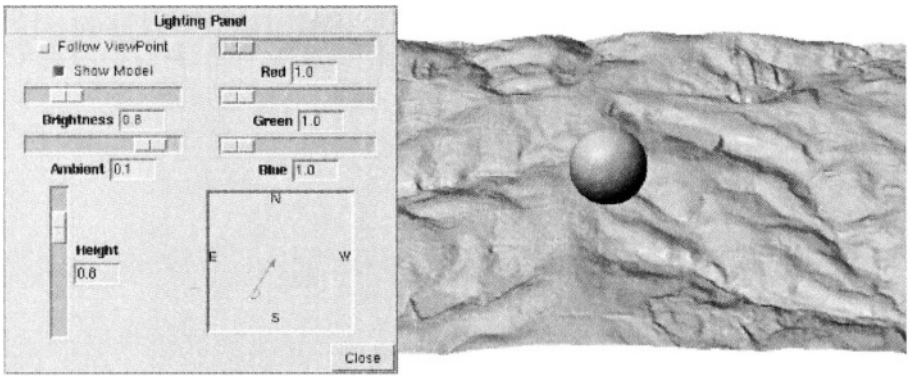


Figure 8.5. Interactive control of light aided by a sphere

Saving images, state and view settings. To save the created image, go to "File" ⇝ "Image dump" and select one of the formats that you want to use (SGI/RGB-Format, TIFF, PPM). You can transform the saved image to JPEG, Postscript or other formats using a graphics program such as `gimp`. To render and save image at high resolution (for example, at 5000 x 5000 pixels), use the "full resolution PPM" option. The image will be split, rendered piece by piece and then patched together.

Using "File" ⇝ "Save state" you can save the settings of your current 3D visualization. You can restore the settings by "File" ⇝ "Load state". Using this capability, you can save and quit your work and then start it again without losing just the right light and view that you have found. It is highly recommended to save the state regularly so that you can go back and restore your settings at later time. You can also save your 3D settings in a GRASS 3d.view file using the "Save 3d settings" option from the "File" menu or load the settings from a previously created file to get specific viewing parameters.

8.2.2 Querying and analyzing data in nviz

To use `nviz` for both qualitative and quantitative analysis you can perform 3D queries. Choose "Panel" ⇝ "What's Here?" and activate the "What's here?" switch. You can select which attributes you want to be included in the query using the button "Attributes". In Figure 8.6 we query the active upper elevation surface with slope map draped over it. By pointing the mouse at a location of interest, get its coordinates including the elevation, the slope value, and a distance from previous queried location. You can also use `nviz` to perform small digitizing tasks in 3D by using "What's here?" and piping the result into a file. The file can then be converted into a site file or a raster

file using `nvizimport` script. The principles used in the algorithm for 3D spatial query are described by Brown et al., 1995. If you have `nviz` compiled with Postgres support, you can directly query connected tables in PostgreSQL DBMS for vector and point data attributes.

Working with multiple surfaces. Multiple surfaces are by default displayed based on their 3D coordinates; however, their relative position can be changed using the position menu, which opens after clicking on the "Panel" ⇝ "Surfaces" ⇝ "Position" button. You can change the relative vertical position of a current surface using the "z" slider or you can move the surface around using the cross in a horizontal position square. Using this interface, you can arrange your surfaces within your graphical window in a way useful for visual analysis; for example, next to each other, as shown in Figure 8.7.

When comparing multiple surfaces, cutting planes can be very useful. Choose "Cutting plane" from "Panel" menu, select a cutting plane (you can have up to 8) and set its appropriate orientation using the "Rotate" slider. You can also slide the rotate slider to interactively cut through your surfaces, or you can slide the "cross" to move the cutting plane through the surfaces in fixed direction. The color of the cutting plane can be set to the color of the top (T) or the bottom (B) surface, blend (BL) of those two colors, grey (GR) or invisible (N) by switching on the appropriate button.

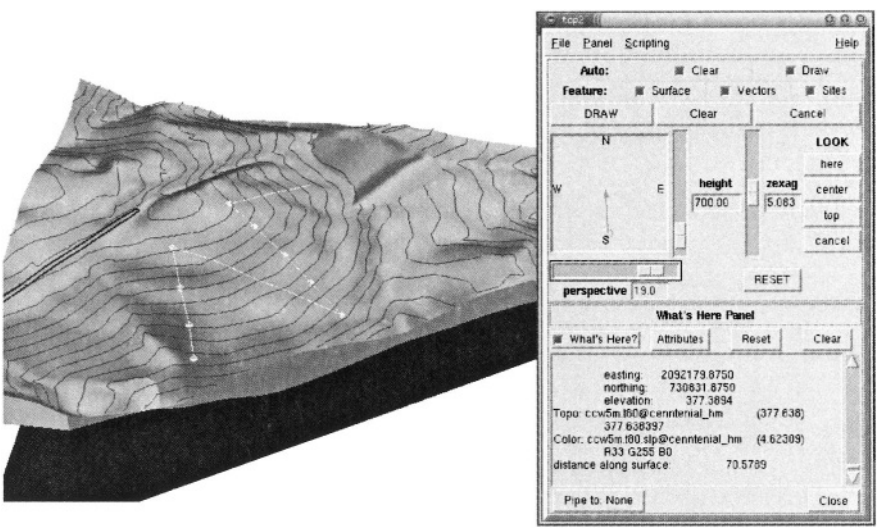


Figure 8.6. Interactive 3D query of elevation surface with slope map draped as color

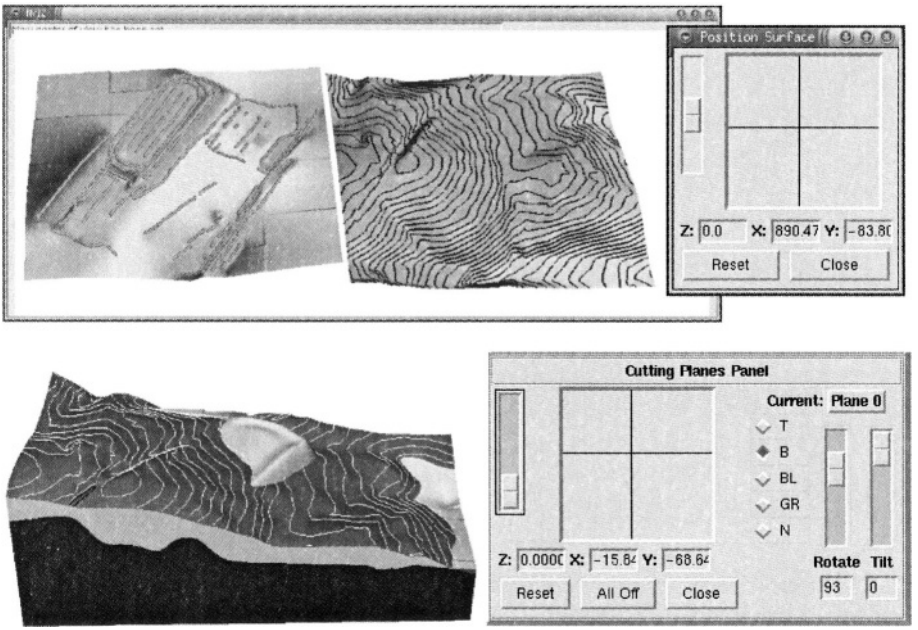


Figure 8.7. Viewing multiple surfaces next to each other or in their relative position with a cutting plane (elevation surface before and after construction)

When working with multiple surfaces and cutting planes, keep in mind that in geoscience applications, the vertical spatial variability requires resolutions much higher than are the resolutions typically used in a horizontal plane. Therefore, a global vertical exaggeration ("zexag") factor is applied for better visual perception of terrain. However, to visualize vertical relationships with sufficient detail, relative exaggeration of depths has to be used. For example, for multiple surfaces representing soil horizons, the depths need to be exaggerated relative to terrain surface (Brown et al., 1995).

8.2.3 Creating animations in 3D space (⇑)

Animation is a powerful tool for exploring large data sets and for analyzing time series observations or modeling results. Fly-by's over digital elevation models or multiple surfaces can be created by using two types of key frame animations. Scripting with file sequence tool provides capabilities to build more complex animations with multiple map layers including combination of surfaces, vector data and sites.

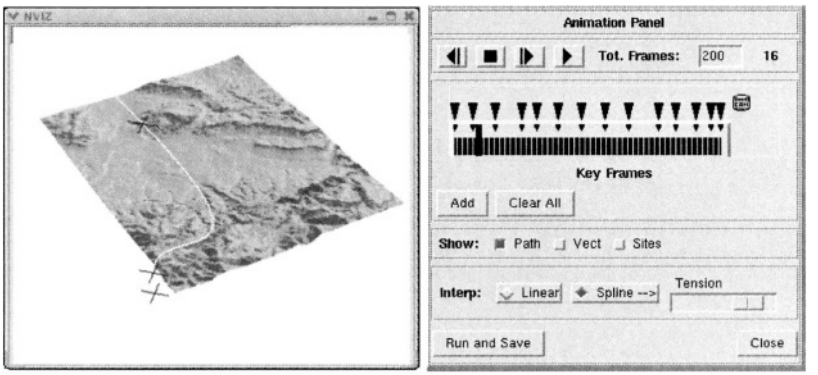


Figure 8.8. Fly-by animation menu in `nviz`

Creating a fly-by. Animations simulating flying over a surface can be created using animation panels which allow you to define *key frames,* representing key positions defining your path and then render and save the surface views along this path. Basic control of the fly-by is provided by the menu "Panel" ⇝ "Animation" (see Figure 8.8). First, enter the number of images to be rendered as "Tot. Frames". The default is 25, a larger number is suggested to get an interesting flight, for example, type in 100 followed by <ENTER>. Now select your initial viewing position using the viewing direction puck and perspective, height or twist sliders in the upper "Controls" panel. Click on "Add" to save this position. Then select the next position on the key frame (time) axis by dragging the thick vertical blue line to the right and define the second key (viewing) position by adjusting the direction, perspective, height or twist. Save your second key frame by clicking on "Add". You can continue moving your viewing position and adding key frames until you reach the end of the time line. Additional frames will be automatically interpolated between the key frames as indicated by the black bars on the time axis.

To avoid a jumpy flight, it is useful to start with a small number of key positions, for example, by locating four key frames after each 25% segment on the time axis. After defining the key positions and a number of key frames, run the coarse resolution movie by clicking on the play button (black right arrow) to get some feeling for how the controls work and see how fast and smoothly you are flying. If the flight is too fast, add more frames. You can also control the number of frames between the key positions by dragging the blue triangle on the "Key Frames" axis. To make the fly path smoother, activate the "Spline" button and control the sharpness of the curves on your path using the "Tension" slider. You can add vector and sites map layers to your surface by switching on the "Show vect" and "Show site" buttons.

After previewing the animation by running the coarse resolution fly-by, you can render the full resolution images for your movie by selecting the "Run and Save" and providing a filename (e.g. `film`). Depending on the resolution, image size, and speed of your computer, the procedure may take some time. The result will be a series of image files, which are automatically numbered (e.g., film00000.ppm - film00099.ppm). The images are then used to create a movie file using external tools as described in the next paragraph.

If you just want to try out animation, the simple animation tools described above are enough; however, for a serious animation work it is worth learning and using the "Key frame animation" panel which provides control over the frame rate and key frame time as well as refined control of the camera (viewing position and direction). The use of "Key frame animation" is described in detail in the *Nviz tutorial.*

Converting series of images to movies. The rendered images are saved in the ppm, tiff or SGI/RGB format. To create an animation, you can merge them into an animation file using the scripts provided with the GRASS code or with external tools such as `mpeg_encode`.[5]

To create an animated GIF using the rendered images `film*.rgb` at half their size run:

```
rgb2gifanim -s 0.5 film.gif "film*.rgb"
```

The MPEG movie can be created in a similar way using the script `make.mpeg`. These scripts use some image processing tools that may not be readily available, but you can create an animation by converting ppm images to gif with `ppm2gif.sh` and then merge them into a single animation file by `gifmerge`. You will find the scripts on the GRASS Tutorials Web site (see the Endnotes for reference).

If you would like to have more control over the creation of the MPEG movie, you can use the MPEG encoding tools directly and define the movie in the parameter file `mpegparam.txt`, for example:

```
OUTPUT my_movie.mpg
INPUT_DIR .
INPUT
#### Put the number of significant digits in [ ]
#### number has to be 5 including the wildcard
#### (max. 99999 images):
film000*.rgb [00-99]
END_INPUT
BASE_FILE_FORMAT PNM
GOP_SIZE        99
INPUT_CONVERT   sgitopnm *
BSEARCH_ALG     CROSS2
PSEARCH_ALG     LOGARITHMIC
```

```
PIXEL            HALF
PATTERN          IBBPBBI
IQSCALE          8
PQSCALE          10
BQSCALE          25
RANGE            10
SLICES_PER_FRAME 1
REFERENCE_FRAME ORIGINAL
```

The parameters can be set based on the documentation for `mpeg_encode`. Of interest are the lines "OUTPUT" (the name of the resulting movie) and the line between "INPUT" and "END_INPUT" (how single frames are located and named). Encoding is performed by running

```
mpeg_encode mpegparam.txt
```

The result may be viewed with MPEG players like `mplayer` or `gtv`. The resulting image size can be controlled using the `nviz` window size or MPEG parameters in "mpegparam.txt". The latter reduces image quality by smoothing while the former is more expensive with regard to CPU and time.

Creating animations using scripting. Complex animations involving multiple surfaces, vector and site data, cutting planes, changing views and light parameters can be created using the scripting capabilities of `nviz`. Animations using dynamic map layers can be used, for example, to view and analyze (Mitas et al., 1997):

- results of dynamic simulations such as water, sediment, pollutant transport, fire spread, migration of animals, traffic, and urban growth;
- time series of observed data from monitoring and remote sensing, such as movement of pollutants, change in rainfall or temperature, past urban growth or vegetation change;
- behavior of a method or algorithm, for example, by animating the results of parameter scans (impact of tension parameter on an interpolated surface, or impact of land cover factor on erosion and deposition pattern, etc.).

You can create scripts for creating animations by using the basic scripting. While the scripting is turned on, the performed visualization tasks are saved in a script file, so that they can be repeated as desired.

Dynamic surfaces can be created using the file sequence tool available under "Scripting" $\rightsquigarrow$ "Script Tools". The use of these tools is rather complex, however, it is well described using the step-by-step example in the *Nviz tutorial.* Many examples of animations created by scripting can be found on the Spatial modeling and visualization web site.[6]

8.2.4 Visualizing volumes (⇑)

Volume data can be visualized in `nviz` using 3D sites and multiple surfaces. Isosurfaces and crossections are being added at the time of this writing, see Figure 8.9. Please refer to the GRASS web site and `nviz` tutorial for the latest update. There are numerous examples of prototype volume and volume-temporal visualizations developed for GRASS using the tools based on SGI GL library at Spatial modeling and visualization web site (see Endnotes of this chapter). GRASS also supports export of volume data in external formats used in other visualization programs, such as `vis5d` (Hibbard et al., 1994).

Coupling with an external OpenGL viewer VisSD. After exporting with `r3.out.v5d` GRASS volumes can be displayed in `vis5d`[7] visualization software. This tool offers various methods to render rotatable semi-transparent volumes, isosurfaces, movable cutting planes and isolines. The code is based on OpenGL which will, such as `nviz`, use hardware acceleration for volume display if a video card capable of hardware OpenGL is used. The software also supports the 3D queries of volumetric data. Figure 8.10 shows an isosurface view of a interpolated volume representing soils pH values. The same volume seen in Figure 8.10 can also be displayed semi-transparent. Refer to the documentation for Vis5D for more details on how to visualize volume data using this tool.

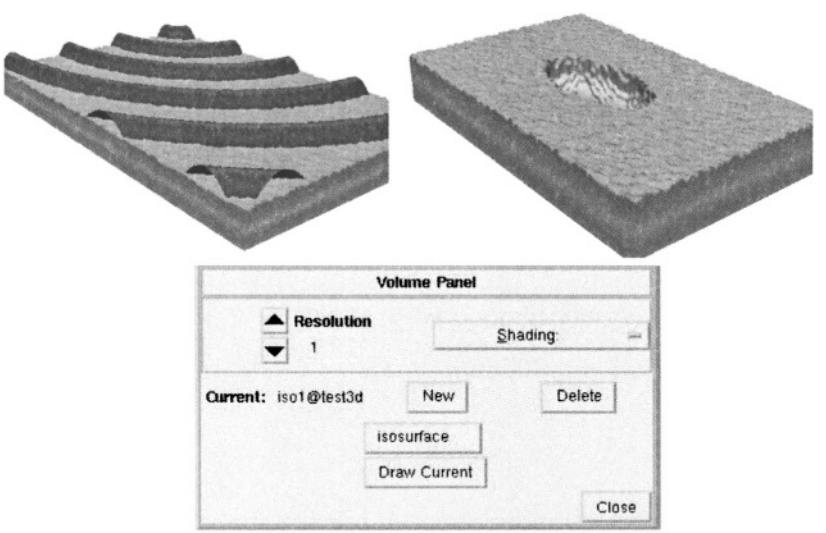


Figure 8.9. Volume (3D grid) visualization integrated in `nviz`: test volumes and a new panel (development and image by Tomas Paudits)

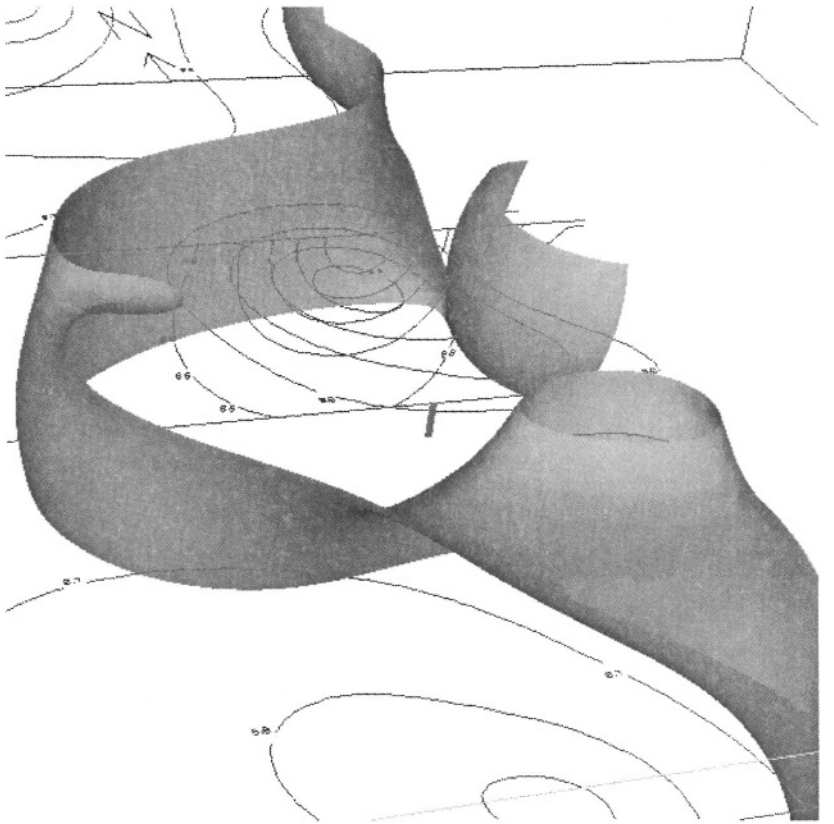


Figure 8.10. 3D pH values displayed in Vis5D visualization tool: isosurfaces and isolines view (Neteler, 2001b)

8.3. CREATING HARDCOPY MAPS

The tools for creating hardcopy maps in GRASS are relatively limited because of its focus on modeling and spatial analysis rather than computer cartography. The hardcopy maps can be created by a text-oriented Postscript graphics tool or in combination with other Open Source graphics programs.

8.3.1 Map generation with ps.map

Hardcopy maps can be created by printing Postscript graphics generated by `ps.map`. The Postscript graphics may be produced interactively or scripted. All raster, vector, and site data are supported, as well as (numbered) grids, user defined icons and a bar scale.

It is useful to run `ps.map` interactively when using it for the first time. After questions for providing the names of the map layers and additional map features, the program asks about saving the instructions to a file. This saved

text file can be used later to apply modifications and to use `ps.map` as a script. Unfortunately, the module still lacks a convenient graphical user interface.

Before starting to produce a map, a printer device defining the paper size has to be selected. As an example we select an ISO-A4 device (before that we list the available devices):

```
ps.select -l
ps.select a4
```

Then you can start

```
ps.map
```

The module will guide you through text menus and allow to customize the map. You may try it with the Spearfish data set and select a raster, a vector and a sites map. Define an appropriate map scale. The module will then ask if you want to save the instructions, we recommend doing that. You can edit this text file later, in case that you would like to make changes and use it in the command line mode of `ps.map`. A sample text file `psmap.def` for a soils map in Spearfish may look like this:

```
raster soils
outline
   color black
   width 1
   end
colortable y
   where 1 6.5
   cols 4
   width 4
   font Helvetica
   end
setcolor 6,8,9 white
setcolor 10 green
vector roads
   width 0.1
   style 0111
   color grey
   masked n
   end
vlegend
   where 4 0
   font Courier
   fontsize 8
   end
text 30% 100% SPEARFISH SOILS MAP
   color red
   width 1
   hcolor black
```

```
   hwidth 1
   background white
   border red
   size 500
   ref lower left
   end
line 606969.73 3423092.91 616969.73 3423092.91
   color yellow
   width 2
   end
point 40% 60%
   color purple
   size 0.5
   masked n
   end
scale 1:125000
grid 2500
   color grey
   numbers 2 grey
   end
end
```

The module is run on command line with these map definitions to produce a Postscript map file:

```
ps.map input=psmap.def output=soils.ps
```

The module generates the Postscript map which may require some time depending on the input map size, the map scale and the selected paper size. Please refer to the manual page to learn more about this module.

After the map is generated, you may preview it with a Postscript interpreter such as `ghostscript` (a convenient graphical user interface for ghostscript is `gv`). If the map is as desired, you can send it to the printer:

```
lpr -s -Pprinter mapname.ps
```

The optional flag `-s` avoids generating another temporal copy of the file in the printer queue. Enter the correct printer name for `printer` (note that there must be no space between `-P` and the printer name).

8.3.2 Map design with Xfig and Skencil

Maps based on GRASS data can be also designed with `xfig`, a general purpose Open Source drawing program with raster and vector support. It is freely available at the Xfig web site.[8] The program can be run interactively and provides full graphical user interface. It reads several raster formats such as TIFF or PNG and supports scale for drawing lines.

To import a raster map, export it from GRASS into a `xfig` accepted format such as TIFF or PNG. In `xfig` the "Picture" icon is used to import and place a raster image within the workspace. For high-resolution raster maps, the PNG driver is recommended (see Section 8.1.3). GRASS vector data can be exported to FIG format with `v.out.xfig`. The resulting file is directly accepted. GRASS sites data may be converted to GRASS vector sites with `s.to.vect`, then exported with `v.out.xfig`. An alternate method is to generate a map with `ps.map` and to convert it to FIG format with `pstoedit`. Read `xfig` documentation to learn more about the use of this tool.

An alternative drawing program to Xfig is Skencil.[9] It is Free Software and supports a range of common vector formats including formats known from the MS-Windows world. The look and feel is different from Xfig. A useful option is support for Scalable Vector Graphics (SVG), and the ESRI SHAPE format which requires the "GeoObjects" add-on. This add-on package is also available from the Skencil web site. GRASS vector maps can be imported in two ways: Either through FIG format using `v.out.xfig` (for details see above) or through SHAPE format using `v.out.shape` (for details on how to export SHAPE data see Section 4.2.3). Vector data will be immediately displayed after loading. Raster data are loaded through the toolbar; on the very right, you will find the "Load raster/EPS" icon. Legends, scales and map frames can be drawn by mouse, using the tools provided by Skencil.

NOTES

1 GRASS 5.3 online user manual, `http://grass.itc.it/gdp/html_grass5/`
2 PNG library, `http://www.libpng.org/pub/png/`
3 Nviz tutorial, `http://grass.itc.it/gdp/online.html`
4 GRASS Tutorials Web site, `http://mpa.itc.it/grasstutor/`
5 MPEG encode software, `ftp://mm-ftp.cs.berkeley.edu/pub/multimedia/mpeg/encode/`
6 Spatial modeling and visualization, `http://skagit.meas.ncsu.edu/~helena/gmslab/`
7 Vis5D software, `http://www.ssec.wisc.edu/~billh/vis5d.html`
8 Xfig drawing software, `http://www.xfig.org`
9 Skencil drawing software, `http://skencil.org/`

Chapter 9

SATELLITE IMAGE PROCESSING

Remote sensing, as a rapidly advancing technology for gathering environmental data using a wide range of satellite platforms, plays a major role in spatio-temporal earth surface monitoring. Throughout this chapter we introduce the basic remote sensing methods and explain their use in GRASS. The tools for image processing and remote sensing applications will be illustrated using a SPOT-1 HRV (Haute Résolution Visible) and PAN (Panchromatic) image data set as well as a LANDSAT-TM7 scene available as an extension for the Spearfish data set.

9.1. REMOTE SENSING BASICS

Before describing numerous methods implemented in GRASS in detail, we will explain basic concepts of satellite remote sensing. As this relatively short section cannot replace related textbooks, references will be given where appropriate.

9.1.1 Spectrum and remote sensing

In principle, there are two different remote sensing approaches: the *optical* (*passive*) and the *microwave* (*active*) systems. Optical remote sensing is based on the measurements of radiation reflected from surfaces. It usually covers the visible (VIS) and infrared (IR) range of the spectrum. The reflected radiation in near (NIR) and middle infrared (MIR) spectrum behaves similarly to visible light, while thermal radiation (TIR) is surface emitted radiation. Longer wavelengths are in far infrared (FIR) range and in the important microwave range. See Figure 9.1 for a portion of the spectrum. The optical region spans at wavelengths from 0.3-15 μm where energy can be collected through lenses.

A subdivision of this optical region is the reflective region, 0.4-3.0 μm (Figure 9.1). The adjacent subdivision of the optical spectral region is the thermal spectral range which is between 3-15 μm, where energy is primarily emitted from surfaces rather than reflected. Far infrared ranges from 15μm-1mm, microwaves from 1mm-1m.

As opposed to optical systems, *radar systems* "actively" emit microwaves and measure the backscattered energy. The major advantage of radar is the relative independence from weather and solar illumination effects. In case of an overcast sky, the earth surface is hidden by clouds for optical satellites. However, radar satellites can continue to deliver usable images since microwaves pass through the cloud cover (this is of special interest in the tropics). Radar analysis is not covered in this book because it is fairly complex. For details please refer to the three microwave volumes by Ulaby, Moore and Fung, 1981, 1982 & 1985, the book by Oliver and Quegan, 1998, or other literature. Additional tutorials are available on the World Wide Web.[1] Besides tools for optical data, GRASS also provides basic capabilities to process radar and thermal data.

In this chapter we focus on images acquired by optical systems because they are widely used and their interpretation as well as data processing is easier than for radar data.

Reflected radiation and atmospheric effects. Optical remote sensing systems are measuring sun energy reflected from earth's surface. While the sun is emitting a full range spectrum with a special energy distribution, only part of the energy reaches the earth's surface. The reason are various absorption and scattering processes within the atmosphere. Figure 9.1 outlines the solar radiation on top of atmosphere and at earth's surface. Before reflected solar energy reaches a sensor, it has passed the atmosphere twice at different angles. Correction of such atmospheric effects often requires image preprocessing. It is important to know that for some ranges of the spectrum, radiation cannot pass the atmosphere at all. Within these absorbed wavelengths optical remote sensing platforms are unable to receive reflected radiation from earth. Therefore, the spectral filters of satellite sensors are defined accordingly. Schowengerdt, 1997, describes these issues at a greater detail.

Remote sensing of the environment considers the sun energy's reflection in the visible and infrared range of the spectrum. Depending on the material of the observed object, the amount of reflected radiation varies. The reflectance curves of three basic materials are shown in Figure 9.2. While water absorbs most radiation in the visible spectrum and all in near-infrared, the radiation reflection of unvegetated sandy soil increases from the visible spectrum to the infrared range. The curve for green vegetation is highly dependent on the contents of chlorophyll. With a small peak for green wavelengths, the overall amount of reflection in the visible spectrum is much lower than for the infrared,

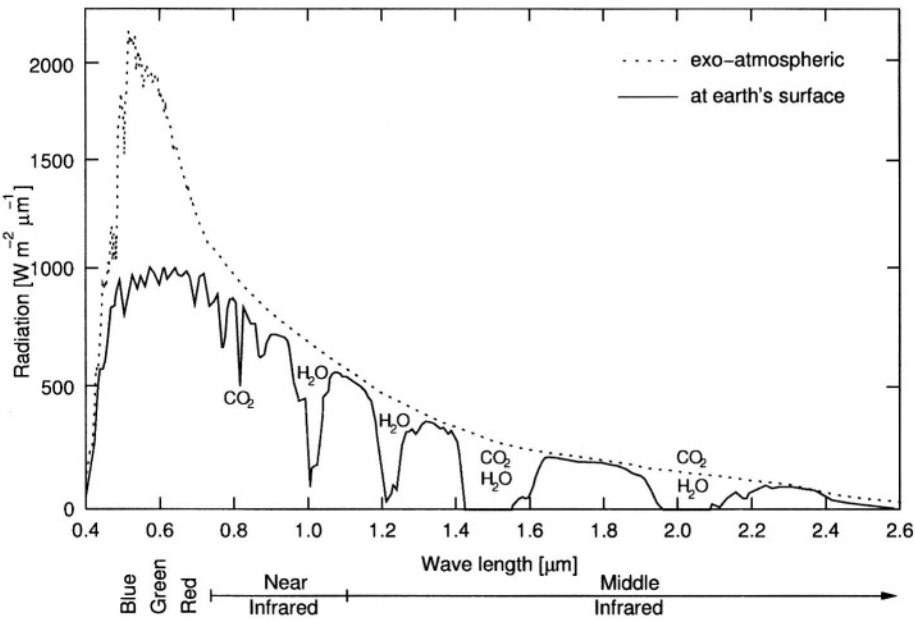


Figure 9.1. Distribution of solar radiation (reflective portion of the spectrum) on upper boundary of atmosphere and at earth's surface with gaseous absorption (solar zenith angle 45°, curves as defined in 6S source code, Vermote et al., 1997)

especially in the near-infrared. The curve depressions at higher wavelength depend on the water contents of the plant. In case of a plant disease or dormant stage the reflection in infrared is dramatically decreased. The different reflectance curves allow us to distinguish the observed objects by multispectral remote sensing. For more theoretical details please refer to Richards and Xiuping, 1999.

9.1.2 Satellite sensors

The payload of the satellite platforms are usually multispectral systems with a narrow spectral range. Some systems are extended by one panchromatic channel which covers a large wavelength range, usually at a higher resolution than the multispectral channels. Figure 9.2 shows the spectral coverage of the LANDSAT-TM5 channels. While the satellite is overpassing the earth's surface, the reflectance radiation is measured line-wise. All satellite sensors are collecting data at the same time, delivering a set of images where every image contains the reflection as it appeared in the related spectral range. Such a data set of geographically related images is called *image scene.* These multispectral satellite data are analyzed statistically as described in this chapter.

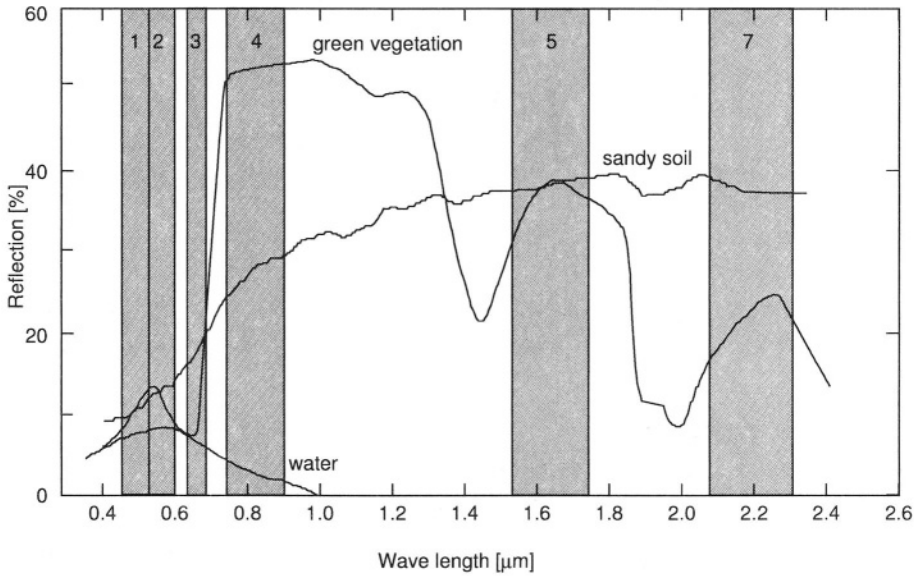


Figure 9.2. Idealized reflection curves of green vegetation, sandy soil and water. For illustration the LANDSAT-TM5 channel filter functions except the thermal channel 6 are shown in background (curves as defined in 6S source code, Vermote et al., 1997)

Details on technical aspects of satellite sensors and their characteristics can be found in Kramer, 1996, Schowengerdt, 1997, and various other remote sensing textbooks.

Besides the different sensor types, satellite systems are also distinguished by their orbit: there are *polar orbiting* and *geostationary* satellite systems. The geostationary orbit is far from the earth (usually at 36,000 km above the earth's surface) which allows the satellite a "static" position of the sensor in relation to the observed area. This orbit is used for weather and telecommunication satellites as they need to always "watch" the same area. Unlike that, near-polar orbiting satellites are rotating around the globe near the poles, covering the entire surface within a fixed range of days due to their own and the earth's rotation. The orbits are near-polar oriented to enable the satellites to cross the earth's surface everywhere at the same time. This leads to a constant solar illumination which is important for multitemporal analysis. Within this chapter we focus on methods for polar orbiting systems.

Resolution. An important aspect of satellite data is the image resolution. In particular, we have to distinguish between:

- spatial (geometric) resolution;
- spectral resolution;
- radiometric resolution.

Spatial or geometric resolution is the spatial extent of each pixel as known from the raster data. For environmental satellite data, this resolution typically ranges from 1 m to 30 m. The sensor-specific geometrical resolution limits the usability of satellite images to certain map scales: Satellites such as the U.S. LANDSAT-TM7, the French SPOT HRV, and the Indian IRS-1D, with a pixel size in their multispectral channels ranging from around 20 m to 30 m, may be used for mapping at a scale of 1:75,000 to 1:250,000. Recently, high spatial resolution satellites with 1 m pixel resolution became operational, however, they provide only a few image channels which limits the number of classes for land use classifications.

Spatial resolution is closely related to geocoding. The satellite image is often distorted and registered in a non-georeferenced coordinate system. Based on *ground control points* (GCPs), the satellite image can be registered to a reference map. This process is called geocoding and is covered later in this chapter. If the satellite image is already geocoded, it can be imported directly into a georeferenced LOCATION.

The spectral resolution refers to the bandwidth of each channel. The bandwidth is the range of the spectrum measured by one channel. Figure 9.2 shows the bandwidths of LANDSAT-TM5 channels with respect to the spectrum and object spectra. Both the higher number of channels and the bandwidth of each channel improve the potential to distinguish objects in an image. For example, when studying crop conditions, phenology-driven signal variations are found in a narrow spectral range between the red and the near-infrared spectrum. Only when this range is covered by two or more narrow bandwidth channels can the effects be studied. Later on we will show how to merge channels with high spatial resolution and with those with high radiometrical resolution to improve classifications.

The radiometric resolution describes the signal dynamics within one image. While the LANDSAT-TM5 channels are effectively measured at 7 bit, other satellites have a radiometric resolution up to 12 bit (e.g. ASTER/TERRA's TIR channels) or even more. For technical details on the various satellite sensors, see the remote sensing textbooks, Kramer, 1996, or Web sites of the space agencies.

In general, a pixel in remote sensing has three parameters: space, wavelength and time (Schowengerdt, 1997:16). Based on these parameters, it is possible to derive thematic maps from satellite data using various methods for image classification.

GRASS provides numerous modules devoted to image processing. Image data are processed using the raster data model; therefore, all raster modules can be applied. To distinguish the specialized image processing tools from the others, they are prefixed with "i." (example: `i.class`).

9.2. SATELLITE DATA IMPORT AND EXPORT

Satellite data are provided in a variety of data formats and sometimes the user can choose according to her/his needs. Common formats are CEOS (unfortunately existing in various subformats), BIL, BSQ, GeoTIFF and others. In general two main format types have to be distinguished:

- formats which contain one channel per file (SUN-raster format, TIFF and PNG format);
- formats which contain one or more channels per file (BIL or BSQ format, CEOS (with various subformats), ERDAS/LAN, NCSA/HDF or HDF-EOS format).

Before working with satellite data in GRASS, you have to verify whether your satellite data are already geocoded because the import method depends on it. This information should be delivered with the metadata belonging to the data set. If there are no metadata present, there might be the chance to retrieve metadata from the image itself using the command `gdalinfo` (which is part of the GDAL library). When specifying the name of the image file as a parameter, `gdalinfo` prints metadata such as projection and boundary coordinates if the image format supports metadata storage.

After a general description of how to build the xy LOCATION, we will start to use the imagery sample data set provided for the Spearfish region. It is available at the GRASS Web site.[2] Beside a SPOT-1 HRV/PAN scene, aerial photos such as NHAP (National High Altitude Photography) multispectral aerial photos and a black-and-white stereo pair with camera information are also included.

9.2.1 Import of raw and geocoded satellite data

Before importing the satellite data, a LOCATION has to be defined. This can be done manually through the GRASS interface or generated automatically with `r.in.gdal`.

Manual definition of a raw data xy LOCATION. Non-georeferenced, (raw) satellite data have to be imported into an xy LOCATION. This procedure is described in Section 3.2. The satellite data set can be imported us-

ing the appropriate module, such as `r.in.gdal` (supports various formats), `r.in.tiff` (TIFF), `r.in.bin` (binary data), `i.in.erdas` (ERDAS/LAN format) etc. For example, to import an ERDAS/LAN file run:

```
i.in.erdas input=landsat5.lan output=tm
```

This imports the `landsat5.lan` data set (a LAN file usually contains several channels) into the GRASS LOCATION. The `output` parameter is a prefix, to which channel numbers are added during import.

Automated creation of a LOCATION. A convenient way to generate a new LOCATION from GIS data or an image data set is to use the module `r.in.gdal`. It is based on the GDAL library and accepts many important GIS and satellite data formats. It optionally allows the user to generate a new LOCATION based on the map metadata from the imported data set. The completeness of metadata depends on the image format, i.e. only some data sets will provide projection information. In terms of raw satellite data, only the xy boundary settings are usually available, so the resulting new LOCATION is generated as xy type. In any case, this method minimizes the efforts to define a LOCATION.

To generate a new LOCATION and to import a raster data set into it, `r.in.gdal` has to be used within another GRASS LOCATION (since the module needs a GRASS environment; as an alternative, see Section 11.3 for a script to generate a LOCATION directly from a GIS raster file outside of GRASS). A new LOCATION is only generated when the `location` parameter is additionally specified. In this case the data stream is written to this new LOCATION into MAPSET PERMANENT. The current LOCATION, where the module was started, is not modified. We also recommend specifying the flag `-e`, which extends a LOCATION if required.

To provide an example, we create a new LOCATION for a LANDSAT-TM7 scene (Spearfish area, 12 July 2002, in UTM/WGS84 projection, GeoTIFF format, available from GLCF Maryland[3]). Since the map datum and ellipsoid do not match the sample Spearfish LOCATION, we cannot import the satellite scene directly. But we can use the Spearfish UTM/NAD27 LOCATION to run `r.in.gdal`:

```
gdalinfo p033r029_7t20000712_z13_nn10.tif
r.in.gdal -e  input=p033r029_7t20000712_z13_nn10.tif \
              output=tm7_2000071.1 location=landsat
```

Now we have to leave GRASS and restart it with the new LOCATION "landsat" to continue the data import for the other channels using `r.in.gdal`. To use this LANDSAT-TM7 scene in the sample Spearfish location, change back to it and run `r.proj` to reproject the channels from the new "landsat" LOCATION into the current LOCATION.

The GeoTIFF format stores the scene projection information, so it will be used by `r.in.gdal` to define the new LOCATION. In other cases, when the satellite data are not geocoded, ground control points (GCPs) are required to reference the image to a reference map. Sometimes, GCPs are provided by the data vendor and stored in the image metadata. If detected during import, they will be written into a so-called `POINTS` text file. This GCPs `POINTS` file can be used later for image rectification. We will explain this in greater detail in Section 9.4.1. Usually GCPs are provided in latitude-longitude coordinates which may not be the desired coordinate system. The optional `target` parameter of `r.in.gdal` allows us to transform the GCPs on the fly to another map coordinate system. The required definitions will be read from the desired georeferenced LOCATION, its name must be given by parameter `target`. An example for a SAR SLC data set in CEOS format:

```
r.in.gdal input=/cdrom/scene1/dat_01.001 output=sar\
                  location=sar_raw target=gauss_boaga
```

This will import the file `dat_01.001` from the mounted CD-ROM into a new LOCATION `sar_raw` with an image prefix `sar` (which in this case will result in the GRASS image names `sar.real` and `sar.imaginary`). The original latitude-longitude GCPs describing the four corners of the image data set are re-projected on the fly to Gauss-Boaga projection as defined in the LOCATION `gauss_boaga` and written to a `POINTS` file in the new LOCATION `sar_raw`.

Because the HDF format is supported by GDAL, geocoded data sets such as ASTER/TERRA and MODIS/TERRA can be imported with `r.in.gdal`.

Installing the Imagery sample data set. For the Spearfish region, an image sample data set is available on the GRASS Web site ("Imagery" package). The installation is done in the same way as for the Spearfish data set. The procedure is described in Section 3.1.3; you just have to substitute the package names. After extraction, you will find a directory `imagery` in your GRASS DATABASE. Start GRASS with the LOCATION `imagery` and specify your name as MAPSET. The following images are available in xy coordinates: `gs13.1, gs14.1` (stereo aerial images from 22. August 1971, camera information is in file `imagery/user1/gscam`), `nhap.1, nhap.2, nhap.3` and `nhap.enh` (multispectral NHAP, National High Altitude Photography[4], a set of aerial infrared image in three bands and a color composite), and a SPOT-1 HRV/PAN scene with channels `spot.ms.1` (green, HRV1), `spot.ms.2` (red, HRV2), `spot.ms.3` (NIR, HRV3), `spot.p` (panchromatic, image scene WRS reference: k=564/j=260, acquisition time: 17:58:50 UTC, date: 27. May 1989). The spectral ranges of SPOT-1 HRV are: 0.50-0.59 μm (green, HRV1), 0.61-0.68 μm (red, HRV2), 0.79-0.89 μm (near infrared, HRV3), all at 20 m spatial resolution. The spectral range of the panchromatic channel is 0.51-0.73 μm at 10 m spatial resolution.

To view the images, open a GRASS monitor and run `slide.show.sh`. Some images are not geocoded, so always use `g.region` with parameter `rast` and the image name to display an image within this LOCATION. Note that this LOCATION is defined with negative "y" coordinates due to historical reasons.

9.2.2 Export of multi-channel data sets

The export of one or multiple channels from GRASS is currently only possible into ERDAS/LAN format using the module `i.out.erdas`. Alternatively you can export single channels into various raster formats, please refer to Chapter 5 for details.

9.3. UNDERSTANDING A SATELLITE DATA SET

The descriptions in this section are based on the SPOT-1 images provided in the "Imagery" data set. Start GRASS with `imagery` LOCATION and your name as MAPSET. The original image data are stored in MAPSET PERMANENT.

9.3.1 Managing channels and colors

A multispectral data set consists of various channels which represent portions of the spectrum. In the case of LANDSAT-TM5 and TM7, the visible spectrum with base colors blue, green and red is mostly covered as well as part of the infrared and thermal spectrum (see above Section 9.1 for spectrum details). Other satellites such as SPOT and ASTER do not provide the blue channel. To visually explore the imagery, we often need to analyze and modify its colors.

Bits, channels and colors. In general, each channel stores the local brightness level pixel-wise at the observed range of wavelength. Each channel thus describes the spectral response pixel-wise within a small portion of the entire spectrum. When storing this information, the satellite sensors are performing a discretization of the continuous signal received from earth's surface into the brightness levels. The number of brightness levels (which is the radiometric resolution) depends on the sensor system. While older sensors like LANDSAT-TM5 internally deliver only 7 bit data ($2^7=128$ grey levels per channel), modern systems capture data at 12 bit or higher ($2^{12}=4096$ grey levels per channel). For data distribution the 8 bit, 16 bit, and 32 bit formats are used. Note that level numbering starts with 0 (no signal, usually colored black). Accordingly, an 8 bit image contains 256 levels numbered from 0 (black) to 255 (white) with different grey levels in between.

The image histogram. The first step in analysis of image data is to look at the channel histograms. Each histogram shows the frequencies of grey levels in an image representing the given channel. For each grey level, the number of pixels in the image is counted and drawn into a diagram. As noted above, the number of grey levels (or brightness levels) depends on the satellite sensor. The x-axis of the diagram represents the grey levels, while the y-axis shows the number of pixels found at that grey level.

To calculate and display the histogram of a channel, open a monitor and run `d.histogram` with the parameter `channel` set to the name of the channel you are interested in. The histogram will be displayed within the monitor using the color coding from the image. If the histogram is displayed in dark colors, consider modifying the image color table (see next paragraph). In the following example, we compare the histogram of the SPOT-1 HRV1 channel and that of an aerial photograph of Spearfish:

```
g.region rast=spot.ms.1
d.mon x0
d.histogram spot.ms.1
g.region rast=gs13.1
d.mon x1
d.histogram gs13.1
```

The histograms are very different: While the distribution of the SPOT-1 HRV1 cell values is highly skewed, the histogram of the aerial photograph slowly decreases over the grey levels.

Color tables. Management of raster map color tables was already discussed in Section 5.1.1. Here, we revisit them with respect to images. Colors are not hard-coded in images but assigned to certain pixel values using so-called *look-up tables* (LUT). The GRASS module for creating LUTs is `r.colors`. Besides the pre-defined color tables like `rainbow` or `gyr` (from green to yellow to red), you can also define your own color assignments (set parameter `col=rules`). After data import, you can change the color table to useful values using `r.colors` with the option `color=grey.eq`. Using this option, a contrast stretch will be applied to the image's color table based on the histogram found for the particular channel. Satellite images are often very dark after import, the contrast enhancement is therefore useful for later data analysis. For example, to improve the contrast of the image channel `spot.p` run:

```
g.region rast=spot.p -p
d.rast spot.p
r.colors spot.p color=grey.eq
d.rast spot.p
```

In this example, we have modified an image in a different mapset which is not allowed for most commands. Here, the operation is possible because

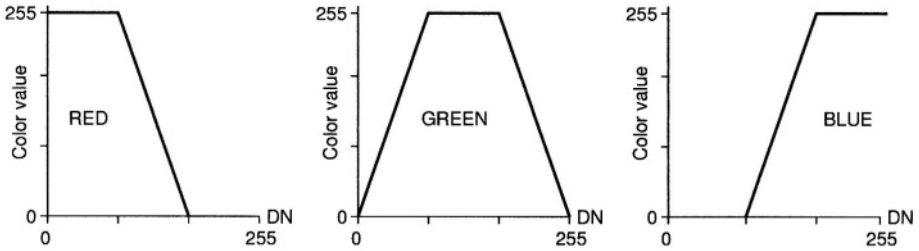


Figure 9.3. Color functions for density slicing of grey scale images (DN: Digital Number)

`r.colors` is able to store a local color table assigned to the original map in the current mapset.

Density slicing. Visual interpretation of grey-scale images can be enhanced by pseudocolor methods that assign various colors to the grey values. One of such methods is *density slicing,* which assigns a range of contiguous grey levels to a range of colors. The full range of grey levels of a channel is cut into three parts and is assigned piece-wise to the base colors blue, green, and red. To perform such density slicing in GRASS, the minimum and maximum pixel values have to be identified (e.g. using `r.info` which shows the range of a map). A color table then needs to be written and used for `r.colors`. The color values are interpolated between the given values and sliced individually between minimum and maximum. For example, one-third of the color range is assigned to red, one-third to green, and one-third to blue, where the overlapping color ranges lead to mixed colors (see Figure 9.3 for details). As an example, we apply a density slice to the panchromatic SPOT-1 image. The range of `spot.p` is 15 - 254; therefore, we slice at the cell value thresholds: 80 (which is (254-15)/3) and 159 (which is (254-15)∗ 2/3), beginning at 15 and ending at 254:

```
g.region rast=spot.p
d.rast spot.p
r.info spot.p
r.colors spot.p col=rules << EOF
15 255 0 0
80 255 255 0
159 0 255 255
254 0 0 255
EOF
d.rast spot.p
```

You may experiment with different ranges for the three colors to optimize the result that is to achieve a good contrast. Also, `d.histogram` is helpful for

evaluating how well the density slice matches an (often) skewed histogram. As two clouds are present in the image, which heavily influence its grey level distribution, you may try to define the maximum at a much lower value and reduce the other thresholds respectively. The individual color values can also be modified for implementation of different slicing functions.

Simple color composites. To obtain a color image from grey-colored channels, several channels have to be combined by assigning each channel to a different base color. For example, to generate a near-natural colored image, the blue, green, and red channels (each only grey-colored) have to be assigned to blue, green, and red color of the GRASS color composition module. Simple examples can be done with the non-georeferenced SPOT-1 HRV data. First, we apply grey scale color tables to the channels; then, we can generate a near-natural color composite with:

```
g.region rast=spot.ms.1
r.colors spot.ms.3 col=grey
r.colors spot.ms.2 col=grey
r.colors spot.ms.1 col=grey
d.rgb b=spot.ms.2 g=spot.ms.3 r=spot.ms.1
```

Note that the order of assignment is unusual for a near-natural color composite. This is due to the fact that the SPOT-1 satellite does not provide a blue channel. The "trick" shown above is a possible workaround; another option is to generate a synthetic blue channel from the existing channels. Due to the assigned standard grey color tables of the input channels, the resulting image appears very greenish. This may be optimized with modifications to individual grey tables. When working with LANDSAT-TM7 data, the first three channels will be assigned to blue, green, and red, as expected from the color model definition.

This composite, as drawn by `d.rgb` into the GRASS monitor, can be stored in a new raster map using `r.composite` or `i.composite` (the latter requires an image group, as will be explained later on).

A false color composite for SPOT-1 can be done in a similar way, but preferably from a histogram-equalized grey scale color tables calculated in advance:

```
g.region rast=spot.ms.1
r.colors spot.ms.3 col=grey.eq
r.colors spot.ms.2 col=grey.eq
r.colors spot.ms.1 col=grey.eq
d.rgb b=spot.ms.1 g=spot.ms.2 r=spot.ms.3
```

This will display a false color image in the GRASS monitor, as we have effectively assigned the green, red, and near-infrared SPOT-1 HRV channels to blue, green, and red display colors. Green vegetation appears red while unvegetated

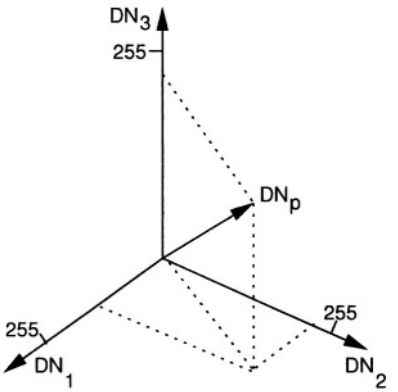


Figure 9.4. Pixel in a three-dimensional feature space with digital number DN_p (adapted from Schowengerdt, 1997:119)

areas are bluish. A geocoded SPOT-1 false color image is already included in the Spearfish LOCATION (`spot.image`).

We provide a more detailed explanation of color models and image composites in Section 9.7.

9.3.2 The feature space and image groups

The processing of data from a multispectral satellite sensor is based on the concept of feature space defined by the sensor channels. Together with the definition of an image group as a combination of multiple channels, this concept is a foundation for image classifications.

The feature space. The channels of a multispectral satellite sensor are considered to span a multi-dimensional coordinate system called feature space. For example, the three channels covering visible light (blue, green, red) span a three-dimensional coordinate system. Within the coordinate system (or feature space), every pixel reaches a certain position depending on the brightness levels in each channel. This position can be considered a vector in the multi-dimensional coordinate system. The brightness levels of the pixels in the different channels are called digital numbers (DN). Figure 9.4 shows the position of a pixel in a three-dimensional feature space, which may represent the blue (DN_1), green (DN_2), and red (DN_3) spectrum range.

The concept of feature space plays an important role in image classifications that are used to derive land use maps from satellite data. Classification methods are based on the idea that pixels containing the same land use are close to each other within the multi-dimensional feature space. The number of dimensions depends on the number of input channels. For example, a number of multispectral pixels which cover a forest will show similar spectral signatures

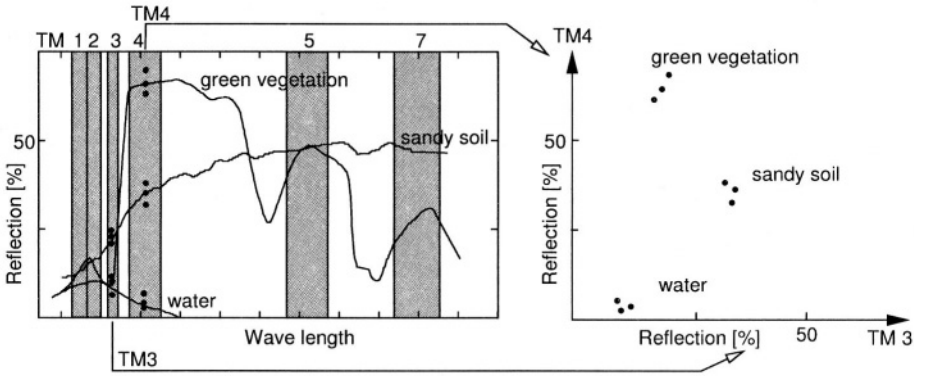


Figure 9.5. Left: Spectrum showing typical spectral response of common objects with LANDSAT-TM5 channels 3 (red) and 4 (NIR); right: Two-dimensional feature space of channels tm3 and tm4 with pixel brightness levels. Three pixels for each observed object (water, sandy soil and green vegetation) are shown with their brightness levels in 3 and 4. The feature space scatterplot (right) represents reflection per channel as appear in channels tm3 and tm4 (adapted from Neteler, 2000:158)

and therefore should be assigned to land use class "forest". In the process of reclassification, many methods (e.g. cluster algorithm) "group" adjacent pixels in the multi-dimensional feature space and assign them to the same class. Figure 9.5 shows the relationship between the spectrum and a two-dimensional feature space spanned by the red and the near-infrared channel of LANDSAT-TM5. More details on image reclassification will be explained below.

For illustration, we display the feature space of two channels of the SPOT-1 satellite scene in the GRASS monitor (red and near-infrared channels):

```
g.region rast=spot.ms.2
dcorrelate.sh spot.ms.2 spot.ms.3
```

The pixel clouds from the red and near-infrared channels span the two-dimensional feature space, which will be partitioned in a reclassification to extract land use classes. However, in the above scatterplot, it is difficult to visually distinguish clustered areas. If you zoom into a small subregion of a SPOT-1 HRV channel and re-run the `dcorrelate.sh` script, the scatterplot will be different, with easier to distinguish pixel clusters. For higher level graphical data analysis external software like "R" (compare Section 13.2) or "XGobi"[5] are recommended.

Image groups in GRASS. Since we are dealing with multi-spectral data sets, we need a method to "bundle" the channels which belong together. This helps when operating on multiple channels with identical geolocation. GRASS

offers a tool to "group" images by selecting the channels: `i.group`. Several multi-spectral image processing modules expect an image group, a few of them even need a subgroup (which may contain the same channels or a subset). Even when working only with a single image, this channel has to be assigned to a group. We are now ready to explore more complex remote sensing issues.

9.4. GEOMETRIC AND RADIOMETRIC PREPROCESSING

In this section, we explain how to preprocess satellite data for further analysis. After geocoding the data set, we continue with radiometric preprocessing to statistically explore the data and to extract further information. To illustrate the applications, we again use the SPOT scene as well as the free LANDSAT-TM7 scene.

9.4.1 Geometric preprocessing

If the imagery data are analyzed in conjunction with other GIS data, geometric preprocessing is crucial. While it may be acceptable to use the image coordinate system for image-only analysis, for a combined image/map analysis and distribution of the results to other GIS users, a high quality geocoding is essential.

Geocoding is related to the terms "ground control points" (GCPs, also called "tie points") and "projection". Ground control points are sites of known position within an image. They are used to perform an image rectification to a reference. These GCPs can either be the four corner points of an image or, preferably, distributed over the image. If available, GPS (Global Positioning System) data can be used as reference points.

General methods of geocoding. Depending on the data provider and the product level, the data set may already contain GCPs or it may be already geocoded. In the latter case, no further geocoding is needed, only potential transformation to the target coordinate system. The following situations can occur when obtaining a satellite data set:

1 the image data set has been already geocoded and projected by the data supplier;

2 the image data set is not geocoded, but contains the four corner GCPs;

3 the image data set is not geocoded, and does not contain any GCPs.

Case 1: the image data set is already geocoded. In this case, you have to verify whether the data are in the target coordinate system. If so, the data

set can be directly imported into the target LOCATION and used for further analysis. If it is in a different coordinate system, the data set must be imported into its own LOCATION and projected to the target LOCATION with `r.proj` (see Section 3.3.2). It is important to check the geometrical accuracy with reference maps.

Case 2: the image data set is not geocoded, but GCPs are present. When importing the data with `r.in.gdal`, existing GCPs are extracted into a GRASS `POINTS` file (used later by `i.rectify`). If the GCPs represent the four corner points, a Helmert (similarity) transformation can be performed to transform the data into a georeferenced LOCATION (first order polynomial transformation). However, as this linear transformation only stretches and rotates the image, it is not very accurate for satellite images which may also contain distortions within the image. Improvements can be achieved by adding more GCPs and using a higher order polynomial rectification method. As mentioned in the import section of this chapter, the extracted GCPs coordinates, which are usually provided as latitude-longitude coordinates, can optionally be projected on the fly to a target coordinate system. This will be explained in greater detail later.

Case 3: the image data set is not geocoded, no GCPs are present. First, GCPs within the image and on a reference map have to be identified. GPS data can be used instead of the points on the map. The geocoding process in GRASS then requires three steps. Create a source LOCATION which contains the satellite data set. Then create a target LOCATION (usually projected) into which the raw data set will be rectified. This target LOCATION contains reference map(s) for GCPs identification. Finally, after a sufficient number of spatially accurate GCPs is identified, the rectification of the input image or image group from the source to the target LOCATION is performed.

Ground control point identification. GRASS provides tools for on-screen identification of ground control points on digital maps and for assigning points from GPS measurements. Four types of image rectification are generally possible: image to image (staying in xy-coordinates), image to raster map (georeferencing to raster map), image to vector map (georeferencing to vector map), and image to keyboard specified coordinates (referencing against known points such as GPS data). As mentioned above, when working with multi- or hyperspectral data, all channels of interest should be assembled into an image group with `i.group`. As an example, we geocode the SPOT-1 HRV data as provided in the LOCATION `imagery`. First, we select the SPOT-1 HRV data in a group (the PAN image is shifted within the LOCATION `imagery` and has to be rectified in an extra procedure):

```
i.group
```

The following screen appears; enter the image group name `spotmss`:

```
LOCATION: imagery          i.group          MAPSET: markus
This program edits imagery groups. You may add data layers to,
or remove data layers from an imagery group. You may also
create new groups

Please enter the group to be created/modified
GROUP:  spotmss_____________   (list will show available groups)
```

Leave the screen with <ESC><ENTER> and accept it with `y`. Now you reach a new screen where you can select the images:

```
LOCATION: imagery       GROUP: spotmss          MAPSET: markus

Please mark a 'x' by the files to be added in group [spotmss]
MAPSET: PERMANENT
__ gs13.1                    __ spot.p
__ gs14.1
__ nhap.1
__ nhap.2
__ nhap.3
__ nhap.enh
__ spot.comp
x_ spot.ms.1
x_ spot.ms.2
x_ spot.ms.3
[...]
```

You can select a channel by writing an `x`; delete a selection by overwriting the `x` character with blank. Leave the selection screen with <ESC><ENTER> and confirm the selection. In the main menu, you can optionally specify a subgroup, which is required for some GRASS image processing tools. For our example, we can leave `i.group` with <ENTER>. Note that you can also use the module on the command line.

Before starting the GCPs identification, the target LOCATION (in our case the `spearfish`) has to be specified with `i.target`:

```
i.target group=spotmss location=spearfish mapset=user1
```

For the graphical identification of ground control points, GRASS provides a few modules: `i.points` to reference to a raster map, `i.vpoints` to reference to a vector map, and `i.points3` (under development), which combines previous modules and `i.ortho.photo`. Also, ortho-image generation from satellite images will be possible when using `i.points3` with `i.rectify3` by incorporating the elevation and a geometrical sensor model.

For our SPOT-1 example, we use the module `i.points` to interactively define GCPs in the GRASS monitor. The GRASS monitor must be open and the

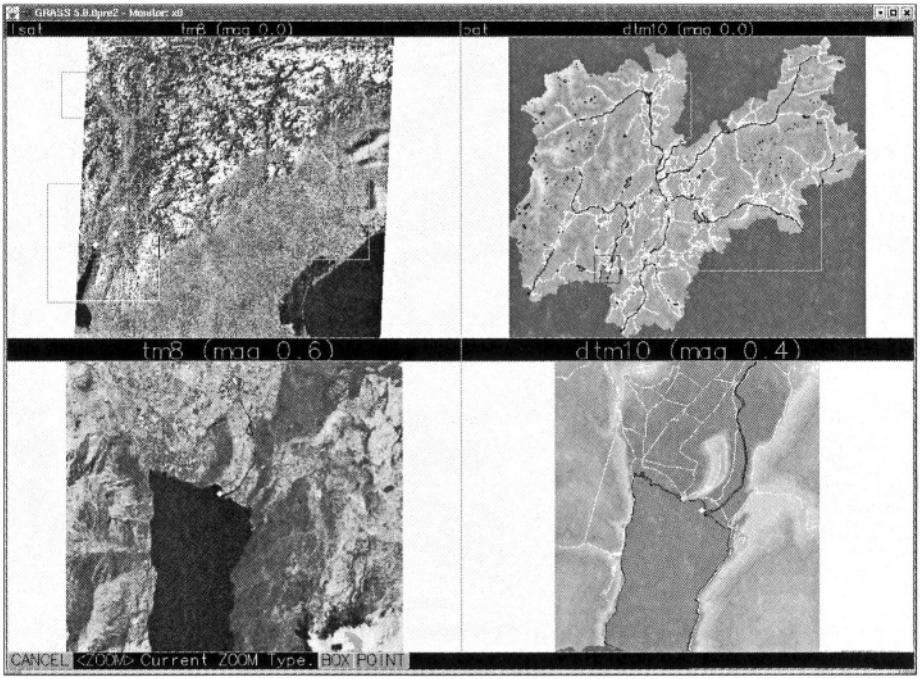


Figure 9.6. Geocoding of a satellite image to raster/vector reference maps with `i.points3`

`i.points` started. It queries for the image group; here, specify the recently created group `spotmss`. An image from the image group can now be selected for display in the upper left half of the GRASS monitor. Select a channel from the list to plot it. This may also be an additional color composite included in the image group. Using the "PLOT RASTER" menu and clicking into the right half of the monitor, a reference map from the target LOCATION can be displayed (for our example, choose the raster map `roads`). Ground control points can now be assigned. It is important to use the "ZOOM" function in both maps to achieve high accuracy. A GCP is set by clicking with mouse at a point, first in the source image, then at the related tie point in the target map. The selection needs to be confirmed and an accepted point pair is drawn in green color.

For the first GCPs pair, you may start and zoom into a road intersection, both in the satellite image and the roads map (select "ZOOM", then "BOX", draw a box with the mouse). The zoomed image/map portions are shown in the lower half of the GRASS monitor (compare Figure 9.6). Remember from vector digitizing that useful points are located in the middle of road intersections, field corners, etc. Alternatively, the input method can be changed to "KEYBOARD" in the menu to directly specify coordinates for a point in the source map in

the terminal window. Using several maps to get the GCPs is recommended; in our example the `fields` map or others can also be used. It is easy to change the displayed channel or reference map during the GCP collection. Figure 9.6 shows a typical screen for geocoding a satellite image in GRASS with `i.points3`.

The "ANALYZE" menu allows us to check the geometrical accuracy of the GCPs. A misplaced point can be disabled (and also enabled again) by double clicking a row in the points table. A disabled point pair is shown in the image and in the list in red. At the bottom of the table, the "RMS error" (root mean square error) is shown; you should try to reach accuracy of half pixel size. For the SPOT-1 example this means the maximum 10 m RMS error when geocoding a multispectral channel with 20 m pixel size. The GCPs are used for all images in the image group; in our example, for the three multispectral channels. When working with `i.vpoints` and referencing against the `roads` vector map, it might be easier to achieve high accuracy.

The number of required GCPs depends on the rectification method. For a linear transformation 4 points are sufficient, preferably selected in the corners of the image. Non-linear polynomial transformations from 2nd to p'th order require more points. The minimum number of points can be calculated as follows:

$$n = \frac{(p+1)\cdot(p+2)}{2} \tag{9.1}$$

where n is the number of required GCPs and p is the order of polynomial used for rectification with `i.rectify`. According to this formula, a third order polynomial transformation (p = 3) requires at least n=10 GCPs. It is recommended to always define more GCPs than the bare minimum. For example, for a third order polynomial transformation, at least 15 GCPs should be identified. Note that for image rectification second or third order polynomials are usually used. It is also important to use GCPs which are homogeneously distributed throughout the image.

The GCPs are stored in an ASCII file `POINTS` within the current LOCATION under a subdirectory `group/<groupname>`. Such a file is written automatically when `r.in.gdal` finds GCPs during the data import in the input data set. In this case, you can check the GCPs accuracy within `i.points` ("ANALYZE" menu entry) which will load an existing `POINTS` file.

Rectification of the image data set. Based on GCPs stored in the `POINTS` file, the module `i.rectify` performs the image rectification into the target LOCATION. Unlike projection and transformation between coordinate systems using `r.proj`, which is based on mathematically defined projection

models, `i.rectify` uses linear or higher order polynomials to warp a source image to a target LOCATION.

After startup of `i.rectify`, the image group has to be specified. For our example we select again the group `spotmss`. The module then asks for the order of polynomials to be used for the transformation ("order of transformation"). The following transformation methods are usually used:

1 Helmert (similarity) = linear: stretching and rotation (needs 4 GCPs);

2 bilinear (affine): allows for stretching and rotation at different scales, able to correct image intrinsic earth curvature (needs 6 GCPs or more);

3 cubic: allows also for earth curvature correction, may tend to overcorrection (needs 10 GCPs or more).

After the order selection, proceed to the next screen with <ESC><ENTER>. Specify new file names for the images that will be stored in the target LOCATION. You may choose the same names or different ones, use `list` to check which names are already in use. After leaving this screen, the module asks for two options:

```
Please select one of the following options
 1. Use the current region in the target location
 2. Determine the smallest region which covers the image
>
```

Both options are used to define the extent of the images in the target LOCATION. The first will clip the image to the current settings and also use the GRID RESOLUTION currently defined in the target LOCATION. Before running `i.rectify`, setting the resolution in the target LOCATION to the image resolution is recommended. For verification or adjustment, you may temporarily restart GRASS with the Spearfish LOCATION and come back to the Imagery LOCATION to continue (for this interrupt `i.rectify` and restart it later). There is, however, an alternative to that: The second option in the above menu allows the user to modify all parameters in a new screen which is similar to the LOCATION definition screen. The default settings are the maximum region extent and resolution from the target boundary coordinates and the image size. You may adjust the values as desired.

After selecting option 1 or leaving this screen when using option 2 (here you are finally asked whether to set the target LOCATION to the new values), `i.rectify` starts to rectify the image(s) in background. This may take a while, depending on the data size and computer speed. The module sends an email when it is finished. After completion of the calculations, the geocoded satellite data are integrated into GIS and may be analyzed along with other GIS data in the target LOCATION. The panchromatic channel `spot.p` needs to be

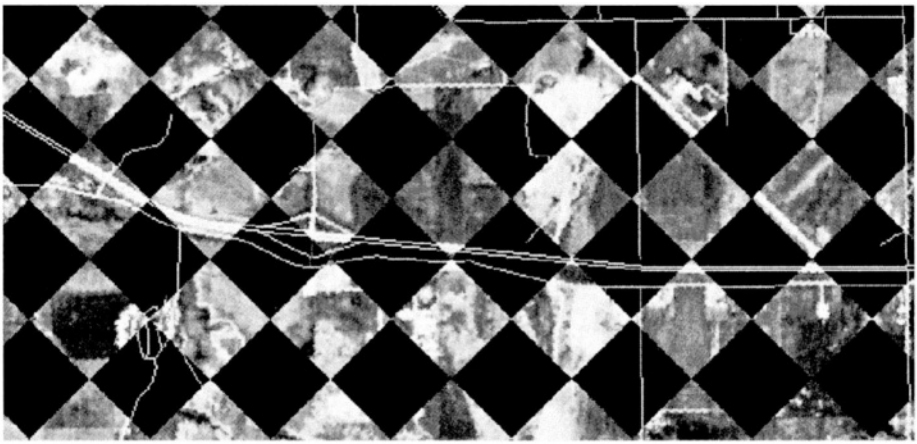


Figure 9.7. Pattern-overlay to verify the geocoding accuracy of a satellite image to a raster reference map

rectified into Spearfish LOCATION using the same procedure as we have just described.

Checking the accuracy. To evaluate the quality of the geocoding process, potential spatial shifts between the rectified channels and the reference map(s) should be verified. The rectified channels can be either displayed in different GRASS monitors or, better, overlayed in small patterns. A pattern oriented overlay method of different data sets can easily be done with MASKs and `r.mapcalc`. We use a rotated square pattern which alternately shows a channel and the reference map within the squares. To generate this pattern, binarized thresholded sine curves are used to generate the MASKs. We display the `roads` map and the geocoded SPOT-1 PAN channel as follows:

```
g.region -p rast=spot.p
#remove MASK if present:
g.remove MASK
v.to.rast roads out=roads10m
#generate binary square pattern mask:
r.mapcalc "MASK=if( (sin(5*row()) + sin(5*col())) > 0.0,\
                                                  null(),1)"
d.rast spot.p
g.remove MASK
#generate inverted binary square pattern mask:
r.mapcalc "MASK=if( (sin(5*row()) + sin(5*col())) > 0.0,\
                                                  1,null())"
d.rast -o roads10m
g.remove MASK
```

The pattern size depends on the current resolution; you may adjust the multiplier to your needs (we used multiplier `5` above). Figure 9.7 shows a portion of an overlay with the roads raster map, resampled from the vector map to 10 m resolution. This approach applies when referencing with topographic raster maps. If the reference map is a vector map, it may be simply overlayed over the rectified image(s) with `d.vect` to verify the geocoding accuracy. If the accuracy is not sufficient, you have to go back to the xy LOCATION and select more or different ground control points. You should make sure that sufficient GCPs are selected and the GCPs are well distributed in the satellite scene.

9.4.2 Radiometric preprocessing

Besides changes to the color lookup tables (LUTs) that are used to enhance the visual perception of an image, satellite data often have to be radiometrically preprocessed so that each pixel represents the apparent radiance measured at the satellite sensor. Up to three major effects have to be corrected, depending on the project goal, the image type, and the observed targets:

- the pixel values are usually a linear transformation of the original data performed by the data provider to fit into the range of 8 bit (0 - 255). By applying "gain" and "bias" (also called "offset") values which are delivered in the image header files or available from the data provider, the DN values (DN: digital numbers) can be recalculated to apparent radiance at sensor values;

- optical data, depending on their spectral range, are influenced by atmospheric effects. To reconstruct the reflectance values at earth's surface (image includes only the values measured at the satellite), each satellite channel has to be atmospherically corrected;

- when the observed target area contains hilly or mountainous regions, the slopes cause variations in the brightness reflectance (terrain effects), which can lead to wrong reclassification results. To overcome this problem, a terrain correction (illumination correction) based on the local slopes derived from an elevation model has to be applied.

Image calibration from DN to apparent radiances at sensor. The gain/bias correction is applied channel-wise to the image data set. The values for the gains and biases are available from the channel headers (leader file) or the data providers.

For LANDSAT-TM7, there are two gain states (low and high gain, see GSFC/NASA, 2001). The rationale behind switching gain states is to maximize the instrument's 8 bit radiometric resolution without saturating the detectors. For thermal data, both low and high gain data are available by default.

For other bands (1 to 5, and 7) the satellite will acquire image data in one of two possible gain settings. High gain measures a lesser radiance range with increased sensitivity over areas of low reflectance. Low gain setting measures a greater radiance with decreased sensor sensitivity for very bright regions to avoid detector saturation.

The leader file of a satellite data set can be analyzed with `gdalinfo` (delivered with GDAL library) for GDAL supported data formats. The program prints important metadata information, including the gain and bias values, if they are present in the data set. Because the original SPOT-1 data are not available, we show an example for a LANDSAT-TM7 data set in CEOS format (first channel):

```
gdalinfo /cdrom/scene1/dat_01.001
 Driver: SAR_CEOS/CEOS SAR Image
 Size is 6920, 5960
 Coordinate System is ''
 Metadata:
 [...]
   CEOS_OFFSET_A0=   -6.2000000000e+00
   CEOS_GAIN_A1=    7.7568627451e-01
   CEOS_GAIN_SETTING=H
 Corner Coordinates:
 Upper Left  (    0.0,    0.0)
 Lower Left  (    0.0, 5960.0)
 Upper Right ( 6920.0,    0.0)
 Lower Right ( 6920.0, 5960.0)
 Center      ( 3460.0, 2980.0)
 Band 1 Block=6920x1 Type=Byte, ColorInterp=Undefined
   Min=0.000/0, Max=255.000/0
```

From this output, we obtain gain and bias (offset) as well as the gain level (high or low gain). The units for gain/bias are usually $\frac{W}{m^2 sr \mu m}$. The general equation for calculation of the apparent pixel radiance at sensor is (Schowengerdt, 1997:313):

$$L_j = gain_j * DN_j + bias_j \tag{9.2}$$

with:
L_j: apparent pixel radiance of channel j $[\frac{W}{m^2 sr \mu m}]$
$bias_j$: offset of linear equation for channel j
$gain_j$: gain of linear equation for channel j

To apply a gain/bias correction, the module `r.mapcalc` can be used. For our example, it will be the following command (we assume, that we have imported the first channel as `tm.1`):

```
r.mapcalc "tm.1rad=0.77568627451 * tm.1 - 6.2"
```

Note that the values depend on the data provider and the image acquisition date as gain/bias values regularly change for various reasons.

With further calculations, it is also possible to convert apparent pixel radiance at sensor to planetary reflectance or albedo (see Mather, 1999:93 and Schowengerdt, 1997:317). These planetary reflectances can be computed to achieve a reduction in between-scene variability through a normalization for solar irradiance. Please refer to the remote sensing literature for details.

Correction of atmospheric effects. Satellite signal distortions are caused by several effects. Diffuse irradiance from sky may increase the radiance of an observed object. Path radiance (atmospheric intrinsic radiance) leads to haze effects. Local effects such as environmental radiance from neighborhood objects change the object's radiance, as well as a locally reduced upward transmittance. Finally, there is the adjacency effect, when a brighter adjacent object influences the surrounding object's radiation. All these problems are widely discussed in the remote sensing literature, see for example Schowengerdt, 1997 (Chapter 2). Atmospheric effects are visible in color composites as a whitish-bluish haze.

The correction of such atmospheric effects is a complex issue. Using an atmosphere model like 6S (*Second Simulation of the Satellite Signal in the Solar Spectrum*[6], Vermote et al., 1997), the radiance at earth's surface can be reconstructed from the apparent radiance at sensor if the local weather conditions at image acquisition time are known. For a method to use the 6S model within GRASS see Neteler, 1999.

As detailed information about the local weather conditions and gaseous contents are often unknown, the atmospheric effects can be retrieved statistically from the image channels themselves. Known dark objects (e.g. water bodies or coniferous forest) can be used to do this. In an unconnected image, these objects do not appear dark due to atmospheric effects. The amount of path radiance is approximately identified by calculating pixel-wise the difference between the actual radiance for a dark object and zero (full absorption, given for water in infrared). This difference value can be removed from all pixels of the channel. Details are described in Moran et al., 1992 and Chavez Jr., 1996. The modules `d.what.rast` or `r.what` can be used to calculate the path radiance for dark objects, the subtraction can be done with `r.mapcalc`. A simpler method, not considered in detail here, is based on the Tasseled Cap transformation. It does not require the manual identification of dark objects and corrects the data set through a "haze" image (Tasseled Cap component TC4) and linear regression. It is implemented in `i.tm.dehaze` for LANDSAT-TM5.

Correction of terrain effects. When observing hilly or mountainous areas, a terrain correction should be applied to the image to correct local brightness

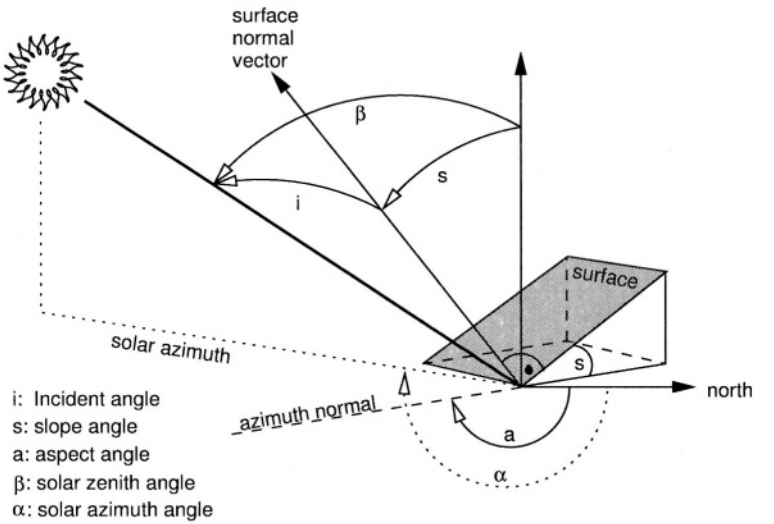


Figure 9.8. Incident angle geometry related to direct solar irradiation onto a tilted surface

changes. These changes result from locally changing incident angles between sun radiation and the slope/aspect of the observed object. A generic approach to normalization of the illumination effects is the "cosine correction" developed by Teillet et al., 1982, and improved, for example, by Sandmeier, 1995. First, the incident angle *i* is calculated for sloped terrain (tilted surface):

$$\cos(i) = \cos(\beta) * \cos(s) + \sin(\beta) * \sin(s) * \cos(\alpha - a) \tag{9.3}$$

with:
i: incident angle between surface normal vector and solar radiation vector [°]
β: solar zenith angle from vertical [°] = incident angle for horizontal surface
s: surface slope angle to horizontal plane [°]
α: solar azimuth angle from north [°]
a: aspect angle of surface from north [°]

The incident angle geometry is shown in Figure 9.8. The sun position data can be retrieved from the metadata of the data set or calculated for a given date and position with `r.sunmask`.

The normalized direct irradiation $E_{\lambda_{dirnorm}}$ for a Lambertian reflector (which equally reflects in all directions) is calculated as follows:

$$E_{\lambda_{dirnorm}} = E_{\lambda_{dir}} * \frac{\cos(\beta)}{\cos(i)} \tag{9.4}$$

with:
$E_{\lambda_{dirnorm}}$: normalized direct irradiation $[\frac{W}{m^2 \mu m}]$
$E_{\lambda_{dir}}$: observed direct irradiation at ground $[\frac{W}{m^2 \mu m}]$

Note that this approach may tend to over-correct the brightness levels for steep slopes. It is also based on the assumption that all observed objects behave like Lambertian reflectors. As this is not true for many land use/land cover types, a modified formula, the Minnaert correction model (besides other models) has been developed. It introduces an empirical Minnaert constant based on the image. We do not go into further details here as this approach is described in the remote sensing literature.

Formula 9.3 and 9.4 can be implemented in GRASS with `r.mapcalc`. You need a slope and aspect map for the image area. Remember that aspects are counted counterclockwise from east in `r.slope.aspect` which requires a reverted orientation and a rotation by 90°. Values for the zenith and azimuth angle of the sun are delivered by `r.sunmask`. Input map is a radiometrically (gain/bias, eventually atmosphere) corrected channel which will be corrected for the terrain effect.

For a simple example, we use the previously geocoded Spearfish SPOT-1 PAN channel (you may also try the SPOT-1 false color composite `spot.image`, which is delivered in the Spearfish data set). First, we need to calculate the sun position to determine the shadow direction and angle. The image acquisition time was 17:58:50 UTC, date: 5/27/89 (see `r.info spot.p`). With `r.sunmask`, we can calculate the sun position. For that, we enter the local time and the time zone:

```
r.sunmask -s y=1989 mon=5 d=27 h=10 min=58 s=50 timez=-7\
                                elev=elevation.dem out=dummy
 Using map center coordinates
 Calculating sun position... (using solpos (V. 11 April 2001)
 from NREL)
 1989.05.27, daynum 147, time: 10:58:50
 long: -103.704231, lat: 44.421085, timezone: -7.000000
 Solar position: sun azimuth 150.233337,
   sun angle above horz.(refraction corrected) 64.488976
 Sunrise time (without refraction): 04:22
 Sunset time  (without refraction): 19:22
 No map calculation requested. Finished.
```

We now have to re-interpolate the elevation model to 10 m to match the resolution of SPOT-1 PAN image to be terrain effect corrected. Then we calculate the incidence angle map according to Formula 9.3. To speed up the calculations we only work in a subregion:

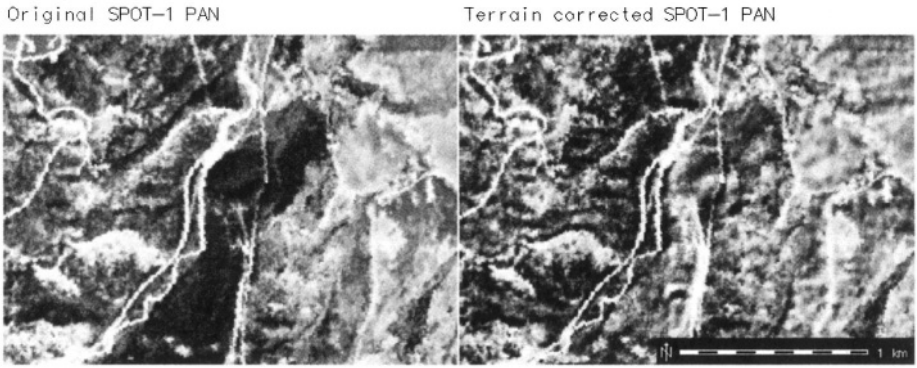


Figure 9.9. Example for cosine correction of terrain effects. Left: uncorrected SPOT-1 PAN image subregion; right: illumination corrected SPOT-1 PAN image subregion (solar azimuth angle: 150°, sun angle above horizon: 64°). Note the area appearing rather dark in the uncorrected image east of the roads. This area is strong inclined with aspect in north-east direction

```
#DEM interpolation to 10m, generate also slope, aspect map:
g.region rast=elevation.dem -p n=4924680 s=4922910\
                                  w=590160 e=592590
r.to.sites -a elevation.dem out=elev30
r.info spot.p
g.region res=10 -pa
s.surf.rst elev30 el=elev10 as=as10 sl=sl10

#generate new aspect map oriented from north, orientation
#clockwise. We are using an internal variable 'as':
r.mapcalc "as10_north=eval(as=360. - as10 + 90.,if(as > 360.,\
                                                   as - 360., as))"

#generate incidence map from sun position:
r.mapcalc "cos_i=cos(90.-64.4)*cos(sl10) + sin(90.-64.4) *\
                        sin(sl10) * cos(150. - as10_north)"
r.colors cos_i col=grey
d.rast cos_i
d.vect roads col=red

#generate contour lines from elev10:
r.contour -q elev10 step=20 out=contour10
d.vect contour10  col=green
d.vect.labels -m contour10 col=green att=cat
```

We apply this incidence angle map `cos_i` along with the sun angle above the horizon as calculated by `r.sunmask` to the SPOT-1 PAN image. This minimizes the terrain effects according to Formula 9.4:

```
r.mapcalc "factor=float(cos(90.-64.4)/cos_i)"
#filter out outlier:
r.mapcalc "factor_filt=if(factor>-5.0 && factor<5.0,factor,1)"
#apply the cosine correction:
r.mapcalc "spot.pcorr=spot.p * factor_filt"
r.colors spot.pcorr col=aspect

d.frame -e
d.frame -c at="0,100,50,100"
d.frame -c at="0,100,0,50"

d.rast spot.p
echo "Original SPOT-1 PAN" | d.text col=white

#select right frame by clicking:
d.frame -s
d.rast spot.pcorr
echo "Terrain corrected SPOT-1 PAN" | d.text col=white
```

The corrected image `spot.pcorr` has reduced topographic effects as shown in Figure 9.9 for a subregion south of Spearfish. Note that cast shadows resulting from relief (mountainous regions) are not taken into account. This approach is only a local correction. It may be extended with cast shadow maps generated by `r.sunmask`. Another simple method to reduce terrain effects is channel rationing which is discussed below.

9.4.3 Application: Deriving a surface temperature map from thermal channel

Several satellites such as ASTER/TERRA, LANDSAT-TM5 and LANDSAT-TM7 provide thermal channels. The data delivered by a thermal channel (channel 6 for LANDSAT systems) can be calibrated to a surface temperature map. These surface temperatures must not be confused with air temperatures. Note that the methods are different for LANDSAT-TM5 and LANDSAT-TM7, as their gain/bias values are different. For an absolute calibration of satellite-derived temperatures, atmospheric correction has to be taken into account.

Surface temperature map from LANDSAT-TM5 channel 6. The following calculations derive the effective at-satellite temperatures (LANDSAT-TM5) of the viewed earth-atmosphere system under an assumption of unity emissivity and using pre-launch calibration constants. In a first step, the gain/bias values are applied to the thermal channel to receive spectral radiances. Then the resulting pixel values are converted to absolute temperature in Kelvin. Optionally it can be recalculated to a degree Celsius temperature map:

```
# convert TM5/b6 digital numbers (DN) to spectral radiances
# (apparent radiance at sensor): radiance = gain * DN + offset
g.region rast=tm.6 -p
r.mapcalc "tm.6rad=1.238+(15.600-1.238)* tm.6 /255."

# convert spectral radiances to absolute temperatures:
# T = K2/ln(K1/L_l + 1))
r.mapcalc "tm.temp_kelvin=1260.56/(log (607.76/tm.6rad + 1.))"

# convert to degree Celsius:
r.mapcalc "tm.temp_celsius=tm.temp_kelvin - 273.15"

# apply new color table, display:
r.colors tm.temp_celsius col=rules << EOF
0 blue
15 green
30 yellow
45 red
EOF
d.rast tm.temp_celsius
d.legend tm.temp_celsius
```

The map `tm.temp_celsius` shows the distributed emitted thermal radiation in degree Celsius. The surface brightness temperature is the actual surface temperature only when the emissivity of the object in a particular waveband equals to 1.0. For most surfaces, where the emissivity is near, but not equal to 1.0, a calibration according to the Stefan-Boltzmann equation is needed when interpreting the results. You may generate a land use/land cover map through image reclassification as shown later in Section 9.8 where you apply the individual emissivity factors according to land use. With `r.mapcalc` you can calibrate the landuse corrected temperature map from these maps. A modified formula for deriving LANDSAT-TM5 surface temperatures was proposed by Singh, 1988 and other authors.

Surface temperature map from LANDSAT-TM7 channel 6. As in the previous case, the LANDSAT-TM7 image data have to be converted from digital numbers to spectral radiances by applying the gain/bias values. Depending on the data format, these values may be retrieved from the image metadata with `gdalinfo` In our example we use the LANDSAT-TM7 scene as prepared in Section 3.3.3 for the Spearfish sample LOCATION. However, the data are provided in GeoTIFF format. This requires to look up the gain and bias parameters from the accompanying metadata file which is separared from the image data.

We use the low gain thermal channel `tm7_2000071.6` as imported earlier (see Section 9.2.1). The conversion procedure is:[7]

```
#rename for convenience
g.rename rast=tm7_2000071.6,tm.6

grep BAND61 p033r029_7x20000712.met
# BAND61_FILE_NAME = "p033r029_7k20000712_z13_nn61.tif"
#  LMAX_BAND61 = 17.040
#  LMIN_BAND61 = 0.000
#  QCALMAX_BAND61 = 255.0
#  QCALMIN_BAND61 = 1.0
#  CORRECTION_METHOD_GAIN_BAND61 = "CPF"
#  STRIPING_BAND61 = "NONE"

# convert TM7/b61 digital numbers (DN) to spectral radiances
# (apparent radiance at sensor)
# radiance = gain * DN + offset
#          = ((LMAX-LMIX)/(QCALMAX-QCALMIN))*(DN-QCALMIN)+LMIN

g.region rast=tm.6 -p
r.mapcalc "tm.6rad=((17.04 - 0.)/(255. - 1.))*(tm.6 - 1.) + 0."

#convert spectral radiances to absolute temperatures:
# T = K2/ln(K1/L_l + 1))
# K1: 666.09 W/ln(m^2 * sr * um)
# K2: 1282.71 Kelvin
r.mapcalc "tm.temp_kelvin=1282.71 /log(666.09 / tm.6rad + 1.)"

#calculate degree Celsius:
r.mapcalc "tm.temp_celsius=tm.temp_kelvin - 273.15"

#apply new color table, display:
r.colors tm.temp_celsius col=rules << EOF
0 blue
15 green
30 yellow
45 red
EOF

#display the map, overlay to ETM/PAN (B80) 15m channel:
g.region rast=etmpan -p
d.his i=etmpan h=tm.temp_celsius
d.legend tm.temp_celsius
```

The resulting temperature map (in degree Celsius) represents the uncorrected surface temperatures at image acquisition time (around 9:30h), see notes above for emissivity correction. For deriving these maps from other satellites such as ASTER/TERRA, please refer to the related documents.[8]

9.5. RADIOMETRIC TRANSFORMATIONS AND IMAGE ENHANCEMENTS

Various methods have been developed for the analysis of multi-channel satellite data using their multispectral nature for radiometric transformations and image enhancement. These techniques play a fundamental role in image interpretation. Most methods may either be applied to uncalibrated data sets or to preprocessed image data sets.

9.5.1 Image ratios

Image ratios are the basis of a simple algebraic method used for feature extraction, reduction of terrain illumination effects, image enhancement, computation of vegetation indices and more (this topic is widely discussed in various papers, for example, refer to Mather, 1999:117-124). To understand a particular channel ratio formula, the object reflectance curves have to be considered (sample curves for green vegetation, soil and water are shown in Figure 9.2). In general, the ratio result for pixels with very different values for the input channels is larger (brighter) than for pixels with similar values. The image ratio equations can be computed with `r.mapcalc`. It is important to include a multiplier of 1.0 at the *beginning* of the map algebra expression because we are dividing integer values. Otherwise, the result will become zero and not the expected floating point numbers. As an example, we calculate a ratio between LANDSAT-TM5/7 channel 7 and 4:

```
r.mapcalc "ratio7.4=1.0 * tm.7 / tm.4"
```

For more than 15 years, a variety of vegetation indices have been developed. To illustrate such a calculation, we can compute a NDVI map (normalized difference vegetation index) from LANDSAT-TM5/7:

```
r.mapcalc "ndvi=1.0 * (tm.4 - tm.3) / (tm.4 + tm.3)"
```

For the calculation of NDVI, the differences between the red and the infrared channel are taken into account pixel-wise to derive information about the vegetation cover. When a pixel value in the near-infrared dominates over the red wavelength (as for green, healthy vegetation), the NDVI is positive. NDVI for unvegetated soil will be slightly above zero; for water, below zero. To quickly classify these three (or more) landcovers, you may filter them with `r.mapcalc` (if-condition).

9.5.2 Principal Component Transformation (⇑)

Multispectral image channels often contain correlations due to similarities of the spectral response of the observed objects or slightly overlapping filter

functions of the spectral sensors. This leads to redundancies within the data set. The "Principal Component Transformation" (PCT) method has been developed to transform such a data set to a new data set without correlations between the channels. This will concentrate the image information in fewer image channels (reduction of image dimensionality), which is of particular interest for hyperspectral data. The PCT transforms the original multispectral data set to a new spectral coordinate system, the Principal Component axes, which are orthogonal to each other. Figure 9.10 shows the position of original multispectral pixels and the PCT coordinate system. In general, the first principal component (PC) image contains the maximum possible variance of the original images. The second principal component image contains the maximum possible variance not stored in the first PC image, as the second PC axis is orthogonal to the first PC axis (Schowengerdt, 1997:191). Accordingly, higher PC images explain remaining variances. The number of PC images is identical to the number of input channels. Since the amount of variance decreases from the first to the last PC, uncorrelated noise (and sometimes some remaining high frequencies) is found in the last PC image. As a results, the method is sometimes used for image compression, as it allows the image information to be concentrated in fewer channels. PCT is also sometimes used to generate additional channels to obtain more variables for a later reclassification process.

The scatterplot in Figure 9.11 shows the original spectral axes and, after transformation, the new rotated PC axes for a sample LANDSAT-TM5 pixel cloud (channel 3 and 4).

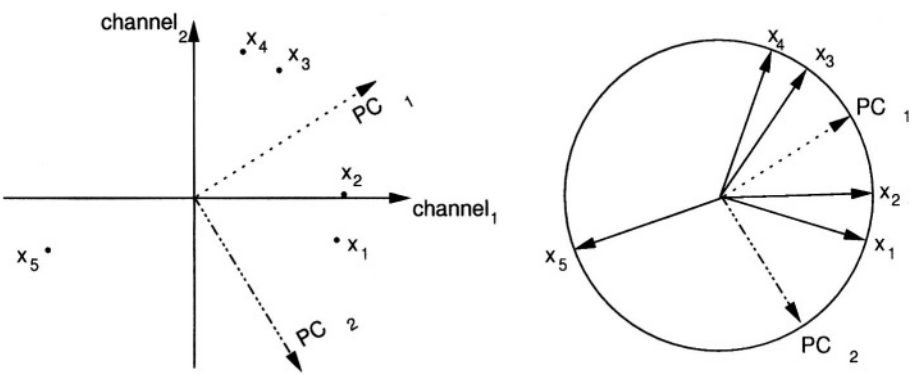


Figure 9.10. Left: Multispectral pixel values shown as standardized data vectors x_1 to x_5 with related first and second orthogonal principal component vectors PCT_1 and PCT_2 in coordinates view. Right: Same data vectors in circle coordinates view

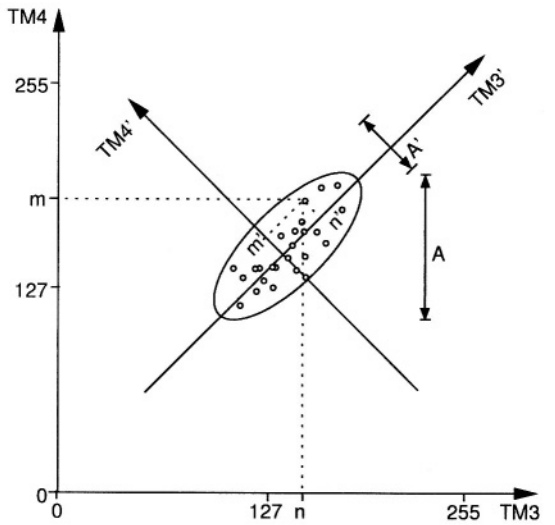


Figure 9.11. Principal Component Transformation applied to channels tm3 and tm4 of a LANDSAT-TM5 data set. Both the original spectral axes (channels tm3, tm4) and the PC axes (PCT transformed channels tm3', tm4') are shown

In GRASS, the Principal Component Transformation is implemented in `i.pca`. The module requires the input channel names (at least two images) and a prefix for the transformed PC image files, which will be enumerated incrementally. Optionally the data can be rescaled to a range different from the default range of 0 - 255.

Another method not covered here is the Fourier transformation, which is provided by `i.fft` and `i.ifft` (forward and backward transformation). It transforms image data from spatial to frequency domain. Among the important applications of the Fourier Transformation are the identification and elimination of (periodic) noise or stripes in a satellite image.

9.6. GEOMETRIC FEATURE ANALYSIS

The geometric (spatial) feature analysis applies local neighborhood operations to raster data. Several methods are available for image smoothing: contrast improvement, low- and high pass filtering, edge detection and more. Geometric filters are user defined raster matrix templates ("moving window") moving row- and column-wise over the image used to calculate the new raster map. All raster cells which are covered by this moving window are considered for the calculations.

9.6.1 Matrix filter: Spatial convolution filtering

Matrix filters can be used to locally modify the spatial frequency characteristics for an image. These modifications are based on calculations considering the neighboring raster pixels in a 2D spatial convolution process (for theoretical details, refer to Richards and Xiuping, 1999:114-116). These spatial convolution filters operate in spatial domain and are an alternative to frequency domain filters (such as Fourier Transformation). Spatial convolution filtering is well suited for:

- high and low pass filtering (sharpening, blurring), averaging;
- edge detection by direction and gradient filters;
- morphological filters;
- preprocessing for image segmentation.

In the next part, we will describe high and low pass filtering and edge detection. In GRASS, two methods are available to define matrix filters. Either the module `r.mapcalc` or the module `r.mfilter` can be used.

The use of `r.mapcalc` is less convenient, as every matrix element has to be addressed with relative coordinates to the central cell. The format to address neighbor cells to the center is `map[r,c]`, where r is the row offset and c is the column offset. For example, `map[0,2]` refers to the same row as the center cell and two columns to the right of the center cell, `map[-2,-1]` refers to the cell two rows up and one column to the left of the center cell. This syntax permits the development of neighborhood-type filters for one single map or across multiple maps. As a simple example, we define a 3x3 low pass filter. The filter equation for the each center cell x_c:

$$x_c = \frac{1}{9}\sum_{i=1}^{9} x_i \tag{9.5}$$

is coded as:

```
r.mapcalc "lowpass=(map[1,-1]+map[1,0]+map[1,1]+map[0,-1]+\
      map[0,0]+map[0,1]+map[-1,-1]+map[-1,0]+map[-1,1])/9."
```

You may try this example with the `spot.ms.1` image. Further examples for `r.mapcalc` matrix operations are described in Shapiro and Westervelt, 1992.

A convenient way to perform spatial convolution filtering is to use `r.mfilter` with a matrix template defined in an ASCII file. We extend our first example to a 7x7 median filter which filters existing sharp contrasts in a raster map. This is effectively a low pass filter. The filter definition has to be stored in an ASCII file, for example, `lowpass`:

```
TITLE 7x7 Low pass
MATRIX 7
1 1 1 1 1 1 1
1 1 1 1 1 1 1
1 1 1 1 1 1 1
1 1 1 1 1 1 1
1 1 1 1 1 1 1
1 1 1 1 1 1 1
1 1 1 1 1 1 1
DIVISOR 49
TYPE P
```

The mean value is preserved if the sum of the filter values equals the line-number * column-number. Every cell within the "moving window" is multiplied by 1. The results are summed up and finally divided by DIVISOR 49 (product of 7 * 7 * 1). To see how this works, the filter may be applied to the SPOT-1 HRV1 channel:

```
r.mfilter spot.ms.1 out=ms1.lowpass filt=lowpass
r.colors ms1.lowpass col=grey.eq
d.rast ms1.lowpass
```

The color table may be set to `grey` or `grey.eq` with `r.colors` as in the example above.

Two types of filters are possible: *sequential* and *parallel* filters. Sequential filters (TYPE S) use the modified neighboring raster cell values for calculation of the central cell, while the parallel filters (TYPE P) use the neighboring cell values of the original map. Directional filters should be set up as parallel filters. Further information related to these types can be found in the manual page for `r.mfilter`.

Another example is a high pass filter for sharpening an image. It can be defined as follows (we store it in an ASCII file `highpass`):

```
TITLE 5x5 High pass
MATRIX 5
-1 -1 -1 -1 -1
-1 -1 -1 -1 -1
-1 -1 24 -1 -1
-1 -1 -1 -1 -1
-1 -1 -1 -1 -1
DIVISOR 25
TYPE P
```

In this example the central cell of the window is weighted by 24, while the other cells have weight -1. The entire matrix is finally divided by 25 and its values are stored in a new map. Again, we apply it to the map `spot.ms.1`:

```
r.mfilter spot.ms.1 out=ms1.highpass filt=highpass
r.colors ms1.highpass col=grey.eq
d.rast ms1.highpass
```

The resulting map shows enhanced high frequencies (at the same spatial resolution). Note that a filter definition file may also contain multiple filters which will be applied to the image subsequently.

The only limitation of `r.mfilter` in comparison to `r.mapcalc` is that only integer numbers are accepted in a filter matrix. If you want to use floating point numbers or trigonometric functions, `r.mapcalc` must be used instead. The latter is also well suited for a thresholded binarization used to extract selected features (if-condition).

9.6.2 Edge detection

Edge detection is an important issue in remote sensing. An edge is defined as a significant change of the pixel values (DN) from one pixel to another. Related filters are also called gradient filters (Schowengerdt, 1997:246-247) such as Sobel, Robert, or other filters. The filter rules to define a Sobel edge detector for `r.mfilter` are shown in the next example. This filter `sobel.filt` is two-fold, as it can operate in north-south and east-west direction:

```
TITLE 3x3 Sobel (edge detection)
MATRIX 3
-1 0 1
-2 0 2
-1 0 1
DIVISOR 1
TYPE P

MATRIX 3
1 2 1
0 0 0
-1 -2 -1
DIVISOR 1
TYPE P
```

To apply it to the `spot.ms. 1` image, we run:

```
r.mfilter in=spot.ms.1 out=ms1.sobel filt=sobel.filt
r.colors ms1.sobel col=grey.eq
d.rast ms1.sobel
```

You can use `r.mapcalc` for a binarization (if-condition):

```
d.histogram ms1.sobel
r.mapcalc "ms1.edge=if(ms1.sobel > 0,1,null())"
d.rast ms1.edge
```

To achieve useful results, additional filtering is required before trying to extract edges. To skeletonize (thin) edges, you can use `r.thin`:

```
r.thin ms1.edge out=ms1.edge.thin
d.rast ms1.edge.thin
```

An application for improved edge detection with vectorization based on segmentation is explained for aerial photographs in Section 10.4.

9.7. IMAGE FUSION

Often, satellite data sets with high radiometric resolution (multispectral channels) lack a high geometric resolution and vice versa. However, for an accurate image interpretation, both radiometric and geometric resolution should be high. Image fusion is a method to geometrically enhance images with high radiometric resolution by merging the multispectral channels with a panchromatic image. Different image fusion methods have been developed; two basic methods will be described in the following sections.

9.7.1 Introduction to RGB and IHS color model

To understand image fusion methods operating in color space, it is important to have basic knowledge about the RGB (red, green, blue) and IHS (also referred to as HIS or HSI: intensity, hue, saturation) color spaces. Similarly to geometrical data the color spaces span their own coordinate systems. Due to their definitions, it is possible to convert images lossless from one color model to the other. The RGB model is an additive color model, where new colors are derived by adding the three base colors at different levels. For example: yel-

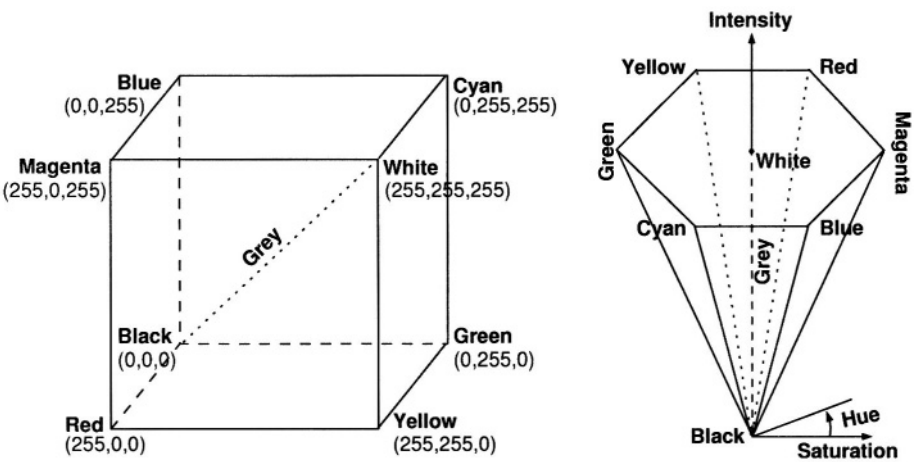


Figure 9.12. Left: RGB (red, green, blue) cubic color space; right: IHS (intensity, hue, saturation) hexcone color space (adapted from Mather, 1999:99)

low = red + green. The IHS model is different; here, the intensity (sometimes also called "value") is a measure of color brightness, the hue corresponds with the dominant wavelength (which is related to color names), and the saturation describes degree of color purity.

Figure 9.12 shows both the RGB and the IHS color model. A pixel in the RGB color space has a specific position within the cube spanned by the coordinate axes, while the IHS color space forms a hexcone. The main advantage of RGB is that it is easy to understand; however, intensity changes are dependent on color settings. Thus, in the RGB model, a change in intensity always leads to a change in colors. The IHS color model preserves colors in case of intensity changes which is a major advantage of this model. Based on this feature, the IHS model can be used to for image fusion, which we explain below. GRASS provides two color conversion modules, the `i.rgb.his` to convert an image from RGB to IHS and `i.his.rgb` to convert back from IHS to RGB.

9.7.2 RGB color composites

Before starting with image fusion in terms of improving the geometrical resolution, we explain the standard color composites. Here three (each grey colored) channels are assigned to the colors red, green and blue; the result is a pixel-wise combined new image with a color table based on the input values as described in Section 9.3.

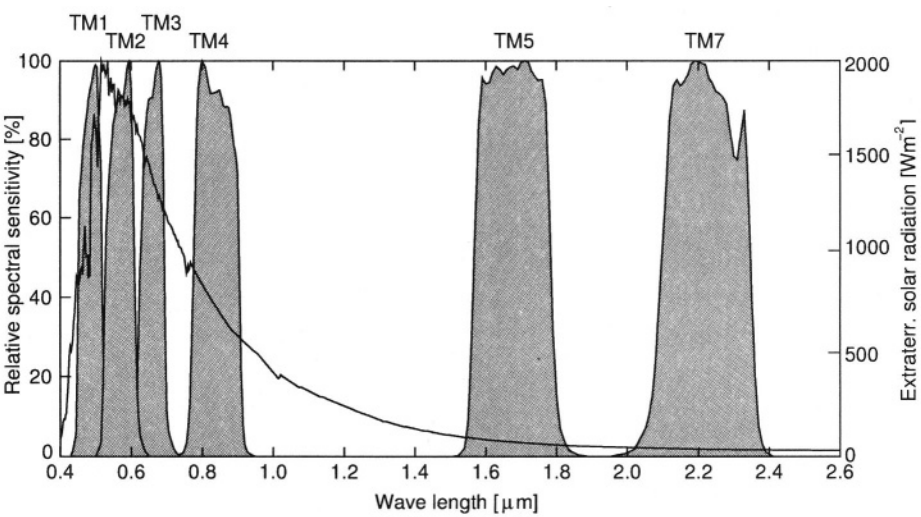


Figure 9.13. Exo-atmospheric solar radiation (in W/m^2 on top of atmosphere, see right axis) and relative spectral sensitivity of LANDSAT-TM5 channel filter functions, thermal channel 6 is not shown (curves as defined in 6S source code, Vermote et al., 1997)

When generating color composites from multispectral data, we need to determine which channels contain most information. For example, when considering LANDSAT-TM5 data, the following problem arises. Figure 9.13 shows the solar spectrum and the filter functions of LANDSAT-TM5. The color filter functions of channels 1, 2, and 3 partly overlap which leads to slightly correlated channels. Generally, 20 color composites can be produced from the six reflective LANDSAT-TM5 channels (not using the emissive thermal channel). Due to the slight correlations, the information contents is reduced when the first channels are combined. A simple method to find out the combinations with the highest information content is the *Optimum Index Factor* method (Chavez et al., 1984), which is based on a correlation analysis. It is implemented in the `i.oif` script. It calculates a rank of all reflective LANDSAT-TM5/7 band combinations and outputs a sorted combination table.

Color composites can be generated with `r.composite` or `i.composite`. The latter requires all channels selected in an image group with at least three channels created using `i.group`. Within the menu of `i.composite`, the selected channels will be assigned to the colors red, green and blue by specifying the letters `r`, `g` and `b`. For example, to generate a near-natural color image, the satellite channels covering the red, green, and blue spectrum have to be assigned to `r`, `g` and `b` respectively. The number of color levels then has to be defined (the number of colors = specified levels^3): e.g., 10 color levels will lead to 1000 colors in the composite image. Due to speed limitations in the current GRASS display color model for large color tables, we recommend not generating 24 bit image composites inside GRASS (use `r.out.ppm3` to export into a 24 bit PPM image). Finally, the output name for the composite has to be specified. After some computation time, the color composite raster image can be visualized in the GRASS monitor. Alternatively, the module `r.composite`, which also operates on command line, can be used.

9.7.3 Image fusion with IHS transformation

For image fusion, two geometrically co-registered data sets are required. The acquisition time of these data sets should be very close to avoid possible modification of the result by land use changes. For IHS-fusion, the three RGB channels must first be transformed to the IHS color model. The general idea of IHS-fusion is to replace the intensity channel with a high resolution panchromatic channel for the back-transformation from the IHS to RGB color model. As a results, the color information in lower resolution is merged with the high spatial resolution of the panchromatic channel. In terms of GIS, a resolution change is required before back-transforming the images to achieve the higher spatial resolution in the output. A disadvantage of this fusion method is that this technique changes the spectral characteristics of the data.

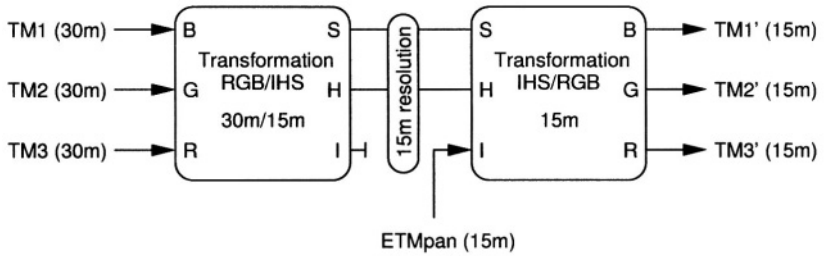


Figure 9.14. Geometric resolution improvement of LANDSAT-TM7 data (IHS image fusion method). After conversion from RGB to IHS color model, the resolution is changed from 30 m to 15 m. The high resolution panchromatic channel replaces the original intensity channel before converting back to RGB color model

As an example, we enhance the geometrical resolution of geocoded LANDSAT-TM7 color channels (each at 30 m resolution) with the panchromatic ETMPAN channel (at 15 m resolution) of the same satellite acquired at the same time. Note that you can also use the SPOT-1 HRV/PAN images we used earlier in this chapter. Before starting the procedure as outlined in Figure 9.14, the input channels should be contrast enhanced with `r.colors`. The input channels are then converted to the IHS color model with `i.rgb.his` at 30 m resolution. Now the resolution is set to the higher resolution with `g.region` as defined by the panchromatic channel. In case of LANDS AT-TM7, it is changed from 30 m to 15 m; for SPOT data, from 20 m to 10 m. To improve the geometric resolution, the original intensity image which resulted from the RGB to IHS transformation is replaced by the panchromatic channel for back-transformation to the RGB color model. Finally, three new RGB channels at 15 m resolution containing the multispectral information from the input channels are generated. The GRASS procedure is as follows:

```
#if not done yet, apply a contrast stretch (histogram equalized)
g.region res=30
r.colors tm.1 color=grey.eq
r.colors tm.2 color=grey.eq
r.colors tm.3 color=grey.eq

#RGB view of RGB channels:
d.rgb r=tm.3 g=tm.2 b=tm.1

#RGB/IHS conversion:
g.region res=15
i.rgb.his red_input=tm.3 green_input=tm.2 blue_input=tm.1\
 hue_output=hue intensity_output=int saturation_output=sat

#IHS/RGB back conversion with ETMPAN replacing
#the old intensity image:
```

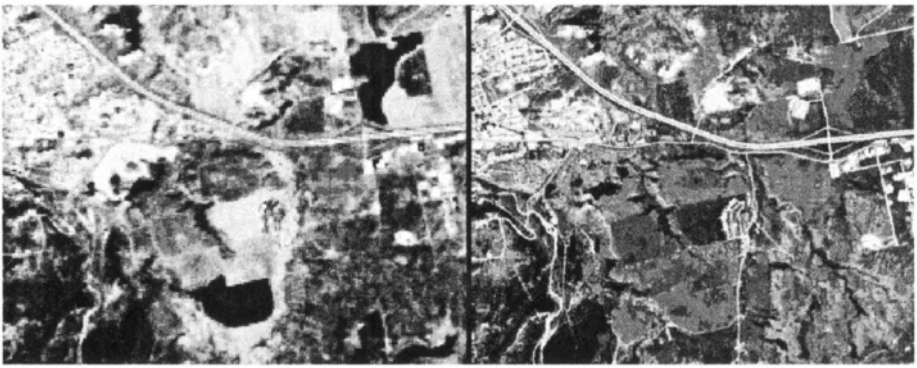


Figure 9.15. Left: Standard RGB composite of SPOT-1 HRV channels (20 m resolution); right: Image fusion of SPOT-1 HRV channels (20 m) with SPOT-1 PAN (10m) with Brovey transformation leading to resolution enhanced image (grey scale reproduction of color original)

```
i.his.rgb hue_input=hue intensity_input=etmpan\
 saturation_input=sat red_output=tm.3_15\
 green_output=tm.2_15 blue_output=tm.1_15

#eventually new contrast enhancement:
r.colors tm.1_15 color=grey.eq
r.colors tm.2_15 color=grey.eq
r.colors tm.3_15 color=grey.eq

#result is are the three color channels with improved
#geometrical resolution:
d.rgb r=tm.3_15 g=tm.2_15 b=tm.1_15
```

More complex merging procedures can also be performed. For geological applications, the use of ratio calculations (generated from `r.mapcalc`) is recommended, as they can be input into the fusion replacing the common multispectral input channels.

9.7.4 Image fusion with Brovey transformation

An alternate method for image fusion is the Brovey transformation (described e.g. in Pohl and van Genderen, 1998, and among other methods in Zhou et al., 1998). The Brovey transformation method can be easily implemented in GRASS. The formula was originally developed for LANDSAT-TM5 and SPOT, but it also works well with LANDSAT-TM7. You need the panchromatic channel to be spatially co-registered to the multispectral channels. Image fusion based on Brovey transformation for LANDSAT-TM7 data merges the channels 2, 4, and 5 (all at 30 m resolution) with the panchromatic ETMPAN channel (at 15 m resolution):

```
g.region res=15
r.mapcalc "brov.red=1. *  tm.5 / (tm.2 + tm.4 + tm.5) * etmpan"
r.mapcalc "brov.green=1. * tm.4 /(tm.2 + tm.4 + tm.5) * etmpan"
r.mapcalc "brov.blue=1. * tm.2 / (tm.2 + tm.4 + tm.5) * etmpan"

r.colors brov.red col=grey
r.colors brov.green col=grey
r.colors brov.blue col=grey
d.rgb r=brov.red g=brov.green b=brov.blue
```

Besides the improved resolution, the result provides a near-natural color table. You may consider modifying the color tables of the resulting channels to optimize the color quality to achieve a near-natural color image.

For SPOT-1 data, the approach described above is slightly modified:

```
g.region res=10
r.mapcalc "brov.red= 1. * spot.ms.3 / (spot.ms.1 + spot.ms.2\
                                      + spot.ms.3) * spot.p"
r.mapcalc "brov.green=1. * spot.ms.2 / (spot.ms.1 + spot.ms.2\
                                       + spot.ms.3) * spot.p"
r.mapcalc "brov.blue= 1. * spot.ms.1 / (spot.ms.1 + spot.ms.2\
                                       + spot.ms.3) * spot.p"
r.colors brov.red col=grey
r.colors brov.green col=grey
r.colors brov.blue col=grey

#note the reversed r/g channels:
d.rgb g=brov.red r=brov.green b=brov.blue
```

Figure 9.15 shows an example for image fusion with SPOT-1 HRV/PAN data.

9.8. THEMATIC RECLASSIFICATION OF SATELLITE DATA

One of the main goals of satellite remote sensing is to derive thematic map layers describing the current land use/land cover of the earth's surface. In a GIS context, these map layers are often used to update maps generated by conventional techniques. Common multispectral reclassification algorithms treat the multi-channel images as variables for a reclassification process. The resulting classes describe the dominating land use or land cover in a certain area, where the land use is considered locally homogeneous. Numerous reclassification methods have been developed; GRASS provides capabilities for a set of standard approaches. Due to its Open Source nature, additional methods may be directly implemented in C programming language.

When reclassifying multispectral satellite data, the image data set is analyzed pixel-wise, with the values of all channels being taken into account for each pixel. The number of pixels covering the same geographic region depends on the number of channels. This group of pixel values describing the same small area is called the spectral vector. It describes its specific position in the feature space (compare Figure 9.5 earlier in this chapter) which contains all spectral vectors. Within the feature space, the reclassification algorithm tries to separate similar spectral vectors which vary depending on the observed object types as soil, vegetation, water bodies etc. Similar spectral vectors will be assigned to the same class. All classes are finally stored in a thematic map where each class describes the dominating land use type.

A problem for the reclassification process are local variations which result from changes within and between the observed objects, slope and aspect changes of the terrain, and variations of the atmospheric conditions (haze, dust, clouds etc.). Depending on the observed area, data have to be radiometrically preprocessed as described in the previous section to minimize influences from slope/aspect and atmospheric effects.

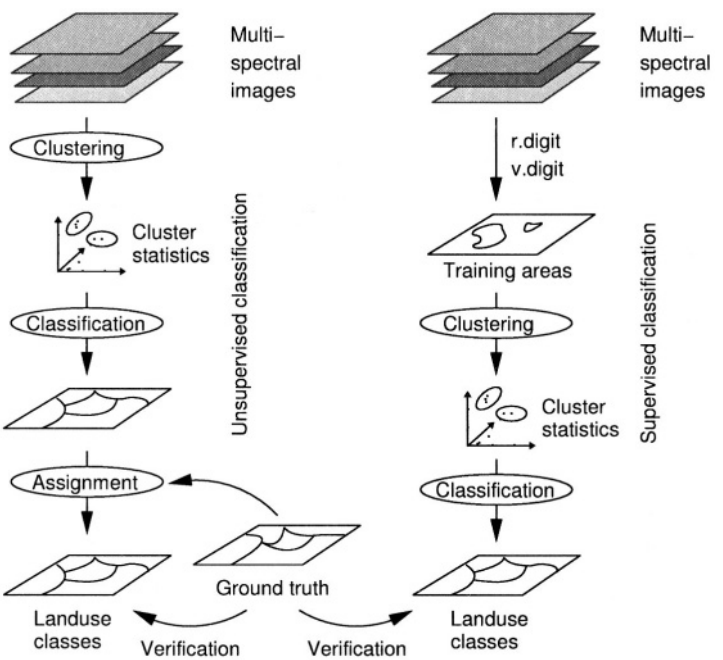


Figure 9.16. Unsupervised (left) and supervised (right) classification procedures for multispectral data

In general, two strategies are common for the reclassification of remote sensing data: *unsupervised* and *supervised* classification methods, which are outlined in Figure 9.16. For both methods, the reclassification process requires two major steps. The data have to be analyzed for similarities in their spectral responses and the pixels have to be assigned to classes. The unsupervised method is fully automated based on image statistics, but it delivers only abstract class numbers. The main task then is to find a reasonable number of clusters/classes and assign ground truth information to these classes. The supervised reclassification requires user interaction, as training areas covering known land use have to be digitized. Image statistics are automatically derived from these training areas and used for the final reclassification.

Common reclassification methods (MLC - Maximum Likelihood classifier as described in Sections 9.8.1 and 9.8.2) are pixel-based. GRASS additionally provides a different method (SMAP - Sequential Maximum A Posteriori classifier, see Section 9.8.3) which also takes into account that neighborhood pixels may be similar. The fact that a neighborhood of similar pixels will lead to spatial autocorrelation is used to improve the result. Altogether, four different approaches for satellite analysis within two main groups of reclassification methods are available:

- Radiometric reclassification:
  - Unsupervised reclassification (`i.cluster`, `i.maxlik` using the Maximum Likelihood reclassification (MLC) method),
  - supervised reclassification and, combined,
  - partial supervised reclassification (`i.class`, `i.gensig`, `i.maxlik`),
- Combined radiometric/geometric supervised reclassification (`i.gensigset`, `i.smap` using the Sequential Maximum A Posteriori reclassification (SMAP) method)

A common problem in remote sensing of the environment are mixed pixels which cover various objects (field borders, urban areas, etc.). In this case, the mentioned methods will assign the pixel dominating object to a class, usually at a low confidence level. If appropriate, masking out settlements where lots of mixed pixels appear may be considered. Subpixel analysis methods such as "Spectral Mixture Analysis" are a way to overcome this problem (for a GRASS implementation see Neteler, 1999).

Other reclassifiers such as Artificial Neural Networks (ANN), k-Nearest Neighbor Classification (kNN), Support Vector Machines (SVM), and other methods are implemented in the R statistical language. They can be linked to GRASS using the GRASS/R interface; for an introduction to `R`, see Section 13.2.

9.8.1 Unsupervised radiometric reclassification

Unsupervised reclassification is the automated assignment of raster pixels to different spectral classes. The assignment is based only on the image statistics. The unsupervised classification is a two-step approach. First, a clustering algorithm groups pixel values with similar statistical properties according to user definitions of minimum cluster size, separability, number of clusters, etc. This approach is similar to the creation of a map legend, where the number of signatures existing in a map is identified and visualized. The pixel clusters are image categories that can be related to land cover types on the ground. The iterative clustering algorithm first computes the cluster mean values and covariance matrices (module `i.cluster`), adjusting these values while reading the image data set. The idea is to identify pixel clouds from the feature space which have similar reflectance values in the various channels. Each pixel cloud, grouped into clusters which represent land use classes, characterizes the spectral signature of a certain object which will be assigned to a class later on.

This cluster information is used to perform the spatial assignment of the individual pixels to the derived clusters (module `i.maxlik`). The MLC determines which spectral class each cell in the image belongs to with the highest probability. Internally, a Chi-square test is run with changing thresholds until a predefined convergence is reached (stability of the pixel assignment during the iteration steps). The result is a new map containing the classes. The MLC also stores the confidence level for each pixel belonging to a certain class in a second map. This map is called "reject threshold map layer" or "rejection map" and contains one calculated confidence level for each reclassified cell in the reclass map. High values in the rejection map represent a high rejection probability for the assigned class. One of the possible uses for this map layer is as a MASK, to identify cells in the reclassified image that have the lowest probability of being assigned to the correct class. It is important to know that MLC assumes that the spectral signatures for each class are normally distributed (i.e., Gaussian in nature) which is often unrealistic. For a detailed discussion, see various remote sensing books such as Mather, 1999.

First step: Clustering of image data. The unsupervised reclassification starts with collecting the image channels of interest (i.e. for optical data usually all reflective channels without thermal channel) into an image group using `i.group`. It is also important to generate a subgroup (menu item 5 in `i.group`) containing the same channels because the classification modules will ask for the subgroup name.

The clustering process is performed with `i.cluster`. A set of parameters has to be specified to control the clustering. It is important to set the initial number of classes used for the first iteration ("number of initial classes"); for other parameters, you may use default values for the first try. They have the

following meaning (class and cluster are used as synonyms, explanations are based on the U.S. Army CERL, 1993 tutorial):

- *Minimum class size:* minimum number of pixels to define a cluster;
- *Class separation:* minimum separation below which clusters will be merged in the iteration process. It depends on the image data being reclassified and the number of final clusters that are statistically acceptable. Its determination requires experimentation, usual values range between 0.5 to 1.5. Note that as the minimum class separation is increased, the maximum number of iterations should also be increased to achieve this separation with a high percentage of convergence (see percent convergence);
- *Percent convergence:* point at which cluster means become stable during the iteration process. When clusters are being created, their means constantly change as pixels are assigned to them and the means are recalculated to include the new pixel. After all clusters have been created, `i.cluster` begins iterations that change cluster means by maximizing the distances between them. As these mean shift, a progressively higher convergence is approached. Because means will never become totally static, a percent convergence and a maximum number of iterations are supplied to stop the iterative process. The percent convergence should be reached before the maximum number of iterations. If the maximum number of iterations is reached, it is probable that the desired percent convergence was not reached. The number of iterations is reported in the cluster statistics in the report file;
- *Maximum number of iterations:* determines the maximum number of iterations which is greater than the number of iterations predicted to achieve the optimum percent convergence of the Chi-square test. If the number of iterations reaches the maximum designated by the user; the user may want to rerun `i.cluster` with a higher number of iterations;
- *Sampling intervals:* simplifies the calculations by grouping the pixels into blocks. If the system resources are about to be depleted due to a too small block size, `i.cluster` will send an email containing a warning. These numbers are optional with default values based on the size of the data set such that the total pixels to be processed is approximately 10,000 (consider round up). With appropriate hardware, the unrecommended sampling may become unnecessary.

After starting `i.cluster`, first enter the group and subgroup names. Then, provide a filename for the "Result signature file" (which will store the cluster information for `i.maxlik`. Only if present (not in the first run), a filename for a "Seed signature" may be specified. This allows you to use cluster information from a previous run or to use spectral signatures from another partial

supervised reclassifications using `i.class`. Then enter a filename for the "Report file" which will be written to the current directory. It contains statistical information about the clustering process. If desired, the module can "Run in background".

The following screen allows us to modify the parameters as described above. For a LANDSAT-TM5/7 scene, the "Number of initial classes" should be initially set to 20. This is a test case – the number has to be changed depending on the results, especially when the convergence is not reached. The other default parameters may be accepted for now. To continue, enter <ESC><ENTER>. Now the cluster analysis is running, generating the cluster statistics and the report file. After checking the quality of the clustering process in the report file, an eventual modification of the parameters and one or more new runs of `i.cluster` are required.

Second step: Unsupervised reclassification of image data. Finally, the unsupervised classification based on the MLC algorithm can be started with `i.maxlik`. The module will assign all pixels in the satellite image to the spectral signatures (classes) derived by the previous clustering process. After starting `i.maxlik`, the image group has to be selected. Then the "Result signature map" which is the result of the clustering process performed with `i.cluster` is queried. Second, a name for the "Classified map layer" (the new reclassified image) which will be created by `i.maxlik` has to be specified. Finally, a name for the "Reject threshold map" is needed to store the pixel assignment confidence levels. As described above this map represents the spatially localized errors which occurred when assigning each pixel to a class. After specifying all parameters, GRASS will compute the unsupervised reclassification. These maps can be displayed now with `d.rast`. The reject threshold map contains one calculated confidence level for each classified cell in the classified image. In case the quality of the reclassification process is not acceptable, the number of classes or other parameters need to be changed subsequently, and the clustering and MLC analysis must be repeated with the new values.

The classes in the reclassification map are then manually assigned to the appropriate land use types in the verification. The assignment of categories can be done with `r.support` ("Edit categories") or `r.reclass`. To change the map colors to more intuitive ones (water colored blue, etc.), the module `r.colors` can be used. A command line based example for reclassifying the SPOT-1 HRV/PAN data into 5 land use classes is as follows:

```
i.group group=spotmss sub=spotmss\
        in=spot.ms.1,spot.ms.2,spot.ms.3,spot.p
```

```
#clustering:
i.cluster group=spotmss sub=spotmss classes=5 sigfile=cluster

#MLC:
i.maxlik group=spotmss sub=spotmss sig=cluster\
              class=mlc.unsup rej=mlc.unsup.rej
d.rast.leg mlc.unsup
d.rast mlc.unsup.rej

#select all areas with confidence level >= 90%
#of correct assignment:
r.report mlc.unsup.rej un=h
r.mapcalc "mlc.unsup.qual=if(mlc.unsup.rej >= 12, 1, null())"
r.report mlc.unsup.qual un=h
d.erase
d.histogram mlc.unsup.rej
```

The filtered rejection map `mlc.unsup.qual` can be used as MASK to select the pixels with a high confidence level of assignment.

9.8.2 Supervised radiometric reclassification

In a supervised reclassification, the classification process is supported by an interactive selection of known areas (for the general workflow see Figure 9.16). Using visual inspection in the field or auxiliary training maps, areas with known land cover are selected and stored in a training map, which is used to identify the spectral signatures for the reclassification process. These known areas are also called "ground truth areas". It is important that the training areas are homogeneous samples. Since training areas cover several pixels, small local variations are included for the definition of the classes. For verification, the module `i.class` supports analyis of channel-wise histograms. A Gaussian distribution of the spectral responses is assumed and standard deviations are displayed in the histograms. These standard deviations can be modified to change the cluster statistics. The spectral signatures (grouped later into classes) are computed from the regional mean values of the training areas and their covariance matrices.

The training areas can either be digitized within the module `i.class` (covered in the first part of the following description) or prepared from auxiliary maps such as already available land use maps (second part of the following description).

Interactive selection of training areas. The manual vectorization of training areas is accomplished with `i.class`. First the satellite channels have to be joined into an image group and subgroup using `i.group`. Creating a natural or false color composite (see Section 9.7.2) which will be helpful for identification of training areas is recommended.

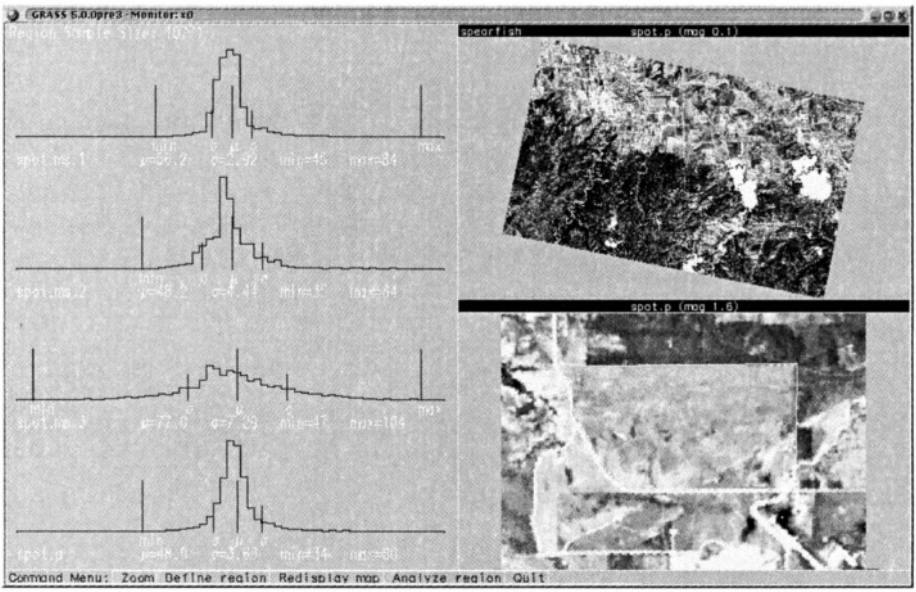


Figure 9.17. Sample screen of interactive training area identification with `i.class` (SPOT-1 PAN image, Spearfish region)

After starting `i.class`., select an image group and subgroup. Then provide a name for the "Result signature file". It will contain the spectral signatures for the later reclassification process. Next is the "Seed signature file" which allows us to read in signatures from a previous run (e.g. in case you interrupted this procedure). Skip it for the first run. Then specify a "Cell map to be displayed" which may be a previously generated natural or false color composite. This map, if not included in the image group, will not be considered for the image statistics. The monitor display becomes divided into three parts. In the upper right corner, you see the image. In the lower right corner, zoomed map portions will be displayed when using the "ZOOM" function. In the left section of the monitor, histograms for the selected training areas for all channels will be displayed. The training areas can be digitized by using the "DEFINE REGION" and the "DRAW REGION" buttons. When digitizing using the mouse, a vector line is drawn around the first training area. Keep in mind that the training area should cover a unique land use. To close the drawn polygon use "COMPLETE REGION" and leave the "DRAW REGION" menu with "DONE". An example screen is shown in Figure 9.17. To verify the cluster statistics, click on "ANALYZE REGION". Now the `i.class` module will search for spectral signatures based on the current training area within the image. The resulting histograms are shown in the left column of the monitor. The

next step is to determine the class assignment to the image – the spatial distribution of the current spectral signatures can be overlayed as filled polygons with "DISPLAY MATCHES" (you can select a color for the area). If desired, the standard deviation can be set to a different value ("SET STD DEV's"). This way, you can try to improve the cluster statistics and display the matches again. After displaying the matching areas, you are asked whether to accept this signature. If yes, you can specify a "Signature description" in the terminal window for this spectral signature. You can then continue to digitize the next training area. With some experience, you will become familiar with the concept of this module. Please note that the vector lines of the digitized training areas are not stored. See the next paragraph for an alternate approach based on retrieving training areas from auxiliary maps.

To obtain good results, you should not digitize border pixels of any land use patch because such pixels often contain mixed spectral signatures. It is also important not to digitize very small areas since these will be ignored for statistical reasons (`i.class` will print a warning message accordingly). Once you leave this module, the generated spectral signatures will be stored.

The spatial assignment of the data set pixels to the classes is done with `i.maxlik`. Specify the file generated by `i.class` as the "Result signature map". The other settings are the same as described in the previous section. Finally, the reclassification map and the "Reject threshold map" are created.

Generating training areas from auxiliary maps. When additional maps with information about the current land use are available, they can be used to extract training areas. It is also useful to digitize training areas independently from `i.class` and store them in a separate map for later use/verification.

For raster maps, `r.mapcalc` (if-conditions) will be useful, for vector maps it will be the `v.extract` module. If training areas need to be digitized from a map, `v.digit` can be used (see Section 6.1.2). After digitizing, vector areas have to be converted with `v.to.rast` to a raster map. It is very important that the training areas are assigned a vector label, for example, within `v.digit`. Otherwise unlabeled areas will not be converted by `v.to.rast`. It is also possible to digitize from a raster image with `r.digit`.

The training map in the raster model is input for `i.gensig`. This module creates a signature file using the training area statistics similar to `i.class`. The spatial assignment of the pixels is subsequently handled by `i.maxlik`.

Partial supervised reclassification. The partial supervised reclassification is similar to the above-described unsupervised reclassification. The difference lies in the incorporation of training areas, which have to be defined prior to the application of `i.cluster` using `i.class` or `i.gensig`. After preparing spectral signatures from training areas, the clustering mod-

ule `i.cluster` is started and the "Result signature file" is generated with `i.class` or `i.gensig` is used as "Seed signature" for `i.cluster`. Finally the `i .maxlik` is used to generate the reclassification map and "Reject threshold map". Beyond this step the procedure is the same as for the unsupervised classification.

Also, in the reverse order, the hierarchical classification is a way to derive thematic maps from satellite data. Based on an unsupervised reclassification, potential training areas are identified and stored in a map. This map is a basis for a supervised reclassification as shown above. As signature files can be used across the modules, better results are eventually achieved through an iterative approach rather than a straight-forward classification.

9.8.3 Supervised combined geometric and radiometric reclassification

GRASS provides an additional sophisticated supervised reclassification tool. The algorithm is a combined radiometric/geometric reclassification method which is called *"SMAP – sequential maximum a posteriori - estimation"*. Unlike the pixel-based approach described above, this method uses an image pyramid approach which also takes neighborhood similarities into account (Schowengerdt, 1997:107, see also Ripley, 1996:167-168). This combination leads to a significant improvement of the reclassification results (Redslob, 1998). A second advantage is that the module also accepts a single channel data, so it can be used for image segmentation which we demonstrate later in Section 10.4. The SMAP implementation module is `i.smap`.

The steps for a SMAP-classification are as follows. First the data set images are joined into a group with `i.group`. Training areas have to be digitized with `v.digit`, `r.digit` or generated from existing vector or raster maps. The training areas map has to be a raster map. Note that the number of training areas defines the number of classes. Similarly to the other reclassification methods, the training areas should cover several pixels; otherwise they will be ignored if the pixel number is too small. The spectral signatures are generated from the training map with `i.gensigset`. This module first queries the name of the training map, then the group and subgroup names. The module creates a "Subgroup signature file" which corresponds to the above mentioned "Result signature file".

From the spectral signatures, the supervised reclassification can be performed with `i.smap`. Again, group and subgroup have to be entered, followed by the name of the recently generated "Subgroup signature file". After this, the computation will be started.

With a sufficient number of training areas, the results of this algorithm are superior to the MLC. A comparison of SMAP, MLC, and the ECHO reclassifier

(the latter is not implemented in GRASS) can be found in McCauley and Engel, 1995.

Summary of the standard reclassification techniques. As explained above, GRASS provides several options to reclassify multispectral data. Table 9.1 summarizes the available main reclassification techniques.

	radiometric, unsupervised	*radiometric, supervised*		*radio- and geometric, supervised*
Preprocessing	i.cluster	i.class (monitor)	i.gensig (maps)	i.gensigset (maps)
Computation	i.maxlik	i.maxlik	i.maxlik	i.smap

Table 9.1. Classification methods in GRASS

NOTES

1 SAR User Guide from Alaska SAR Facility,
`http://www.asf.alaska.edu/SciSARuserGuide.pdf`
Remote Sensing Core Curriculum (RSCC),
`http://www.research.umbc.edu/~tbenjal/umbc7/santabar/rscc.html`
ISPRS tutorial collection,
`http://www.isprs.org/links/tutorial.html`

2 Imagery data set, `http://grass.itc.it/data,html`

3 GLCF Maryland LANDSAT Data for Spearfish (SD) region,
`ftp://ftp.glcf.umiacs.umd.edu/glcf/Landsat/WRS2/p033/r029/`

4 NHAP documents,
`http://edc.usgs.gov/products/aerial/nhap.html`

5 Xgobi/Ggobi software, `http://www.ggobi.org`

6 Atmosphere model 6S (Msix) software,
`http://www-loa.univ-lillel.fr/informatique/system_gb.html`

7 Generating a surface temperature map from LANDSAT-TM7 channel 6, see Landsat 7 Science Data Users Handbook,
`http://landsat7.usgs.gov/resource.html`

8 Asterweb (ASTER/TERRA), `http://asterweb.jpl.nasa.gov`

Chapter 10

PROCESSING OF AERIAL PHOTOS

Aerial photography provides a common base for large scale mapping. It has been widely used for creating and updating maps as well as for maintaining up to date GIS databases. Aerial photos can be used to extract georeferenced data representing topography, landforms, vegetation cover as well as man-made features. Besides creation of thematic maps, area and distance measurements at a large scale (in comparison to satellite-based methods) are often performed. Digital processing of aerial photos in GRASS allows the user to incorporate them into a GIS database. To minimize distortions, mainly displacement due to relief and airplane attitude, orthophotos are generated from (scanned) aerial photographs using a digital elevation model and a referenced map. They combine the characteristics of an aerial photo with the geometric qualities of a map, showing all objects in their precise geographic position.

In the first part of this chapter we introduce generation of orthophotos, in the second part we explain image segmentation of aerial photos for land use classification and edge detection.

10.1. BRIEF INTRODUCTION TO AERIAL PHOTOGRAMMETRY

Before going into details of generating a digital orthophoto, we describe the basic terminology used in aerial photogrammetry. Aerial photos are usually taken with overlap of around 60% for stereoscopic analysis; however, this method is not covered here.

This chapter focus is on vertical aerial photos; however, a brief description of oblique aerial photos processing is included in Section 10.3.3. The aerial photo geometry is shown in Figure 10.1. Assuming that the aircraft which takes

the photos moves (in the ideal case) over the target area without any deviation, the plumb line from the camera will be perpendicular to the horizontal ground plane. This case is called a true vertical image. The plumb line is also called the *optical axis.* Up to deviation of 3° in each direction, the image is called a tilted vertical image. If the inclination exceeds 3°, it is referred to as an oblique image. The nadir (also called plumb point) is the point on the earth's surface (datum plane) which lies exactly below the recording camera. If a photo is taken without deviations from the plumb line, then the nadir will be in the image center (principal point).

If the aircraft was tilted (oblique photo) then the principal point is not identical with the nadir. The principal point is found by connecting opposite fiducial

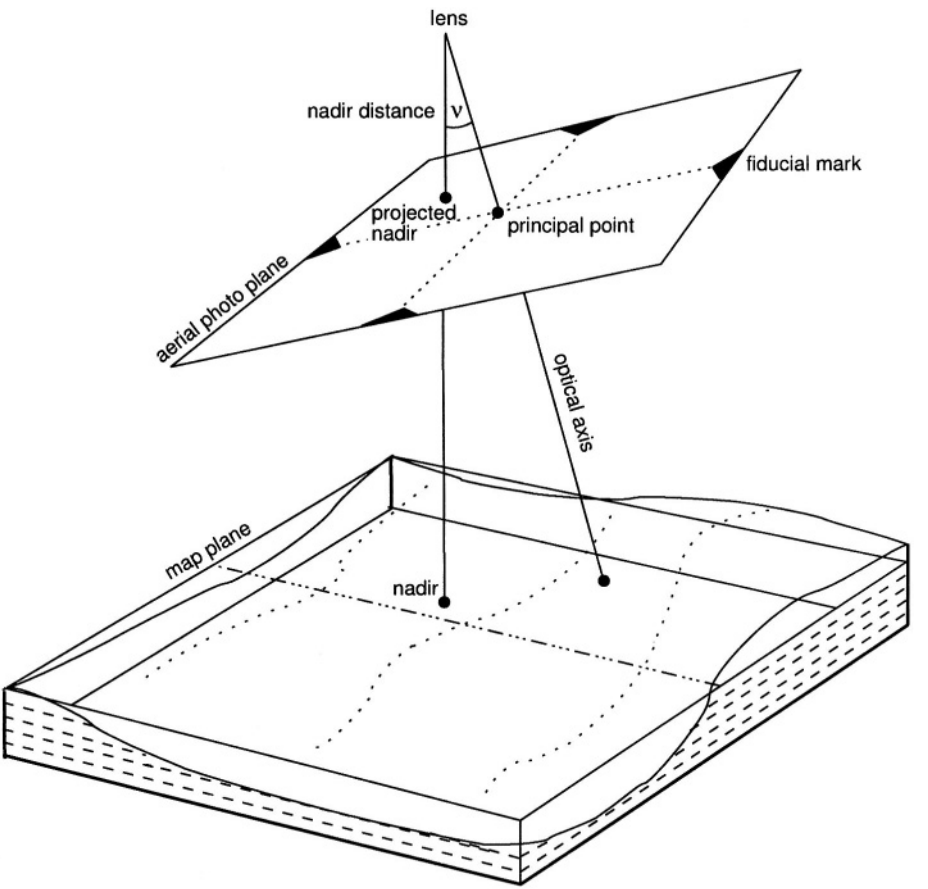


Figure 10.1. Aerial photo terminology (adapted from Neteler, 2000:178)

marks through lines, it is the point where these lines intersect. For oblique photos the nadir is offset from the principal point, the angle between the two being called the nadir-distance (see Figure 10.1). In such case the nadir can be derived by connecting the alignments of vertical objects such as trees or buildings.

A vertical aerial photo is in central projection. This projection can also be assumed for the above mentioned slightly tilted vertical photos. There are significant differences between an aerial photo and a map (which is in orthogonal projection). Because of central projection aerial photos are free from displacements only in the nadir. Displacements increase from the nadir towards the image edges (Hildebrandt, 1996:151), which requires ortho-rectification. These displacements are intensified by uneven topography (see Figure 10.2) or increased aircraft tilt.

Elements of an aerial photo. In an analogous aerial photo metadata are displayed in the annotation bar (instrument strip). It is usually a side bar that includes information about the altimeter, aircraft attitude, watch, approximate focal length and photo counter of the camera. The camera calibration certificate describes the calibrated focal length of the camera that was used to take the photos. It also describes measured distortions within the camera, and the center of symmetry, the principal point (PP). The provider of the photo should be able to make this report available to the user. Ideally, the principal point should fall directly on the intersection of the radii at the center of the picture, which is usually not the case. The small deviation on the order of a few micrometers is relevant for a precise orthophoto. Additionally, the fiducial marks numbering scheme should be given in a diagram (Fig. 10.4 shows examples).

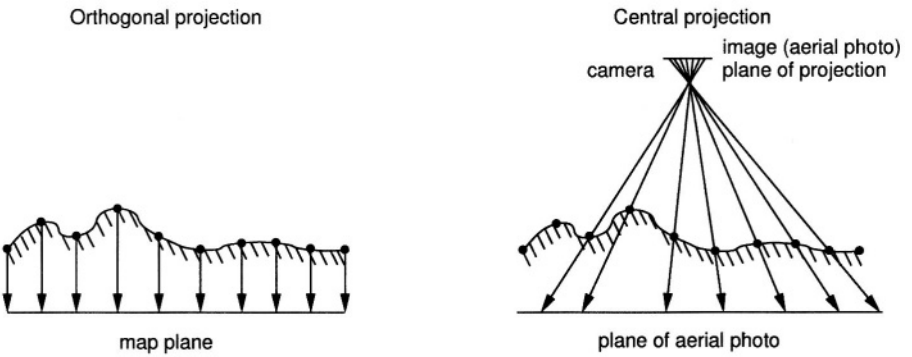


Figure 10.2. Terrain mapping to map plane (orthogonal projection) and aerial photo plane (central projection) (adapted from Albertz, 1991). Objects are displaced in the aerial photo (right) due to relief impact

Only if the report is unavailable, standard camera parameters may be taken from literature, if focal length or camera type are known (e.g. Hildebrandt, 1996:80).

Aerial photos and map scale. The averaged image scale m_b (or the image scale number M) is elevation-dependent and can be calculated as follows:

$$M = \frac{h_g}{f} \tag{10.1}$$

$$m_b = \frac{1}{M} \tag{10.2}$$

The height above ground h_g [in meters] is a function of the topography, the terrain elevation above sea level has to be subtracted from the aircraft's recorded altitude above sea level. The focal length f of the camera is printed approximated (in millimeters) on the aerial photo, or, preferred, given correctly in the calibration certificate. The value must be changed to meters for the formula above. M is the image scale number. The image scale m_b which we are interested in can be derived from Equation 10.2. In general, higher terrain elevations within the aerial photo are recorded in larger scale than lower terrain elevations (Hildebrandt, 1996:152). Furthermore, the horizontal displacement depends on the elevation according to the central projection. Therefore, Equation 10.2 can be used to obtain only an averaged scale for the aerial photo. During ortho-rectification, the displacements are eliminated as the terrain undulation is taken into account.

The image scale is usually not constant over the covered area: If the aircraft was horizontally tilted, then the image scale of the down-side (whose distance to ground is decreased) will be larger than the scale of the up-side. Undulated terrain adds further bias on the overall scale. If the terrain undulation covered by the aerial photo is relatively small it is possible to use an unrectified aerial photo for area or distance measurements. In such a case, minor distortions are ignored (Bierhals, 1988:91). The unrectified photo can be used, if the elevation range in this photo is less than 1/500 of the image scale number. For example, if an aerial photo was taken at a average scale of 1:10,000, the photo may be used unrectified if the elevation range does not exceed 20 m (10000/500=20). Otherwise, an orthophoto should be generated as described below.

The earth's curvature is another factor that affects aerial photos. It becomes a relevant factor only for high altitude or orbit-based images (such as for images from the Russian KVR1000 or similar systems). Note that also shrinking and stretching of film material may cause displacements as well as the usage of a non-photogrammetric scanner.

10.2. FROM AERIAL PHOTO TO ORTHOPHOTO

To generate an orthophoto we need a digital elevation model (DEM) and a topographic reference map. The elevation model is required to normalize the terrain undulation. The raster resolution of the DEM should be similar to the resolution of the aerial photo. Usually such high-resolution DEMs (raster cell length below 1 m) are not available. Therefore, interpolation of a given DEM to a higher resolution, as shown in Section 5.3.4 and Section 7.3.1, is recommended to minimize displacement effects. The topographic reference map is needed to find corresponding ground control points (GCPs) between aerial photo and this map for geocoding. The map scale of the reference map should at least match the average scale of the aerial photo (for example 1:5000), but should be probably larger to provide more details for improved geocoding accuracy. The ground control points are required to register the image xy coordinate system to a georeferenced coordinate system (like UTM or Gauss-Krüger).

Metadata like time stamp, flight altitude above the sea level, tilt, and focal length will be taken from the calibration certificate and the annotation bar. An important requirement are the camera parameters.

In general "true" orthophotos and "pseudo" orthophotos are distinguished. In true orthophotos all elevated objects such as building roofs are corrected for their displacement due to the central perspective. This means that displacement-corrected buildings are seen from above which leads to no-data areas where the displaced roofs have been hiding the ground (on the buildings' back sides as seen from the nadir of the aerial photo). To create true orthophotos, you need an elevation model containing all individual building heights. High resolution DEMs can be produced from stereo measurements of the original aerial photo stereo pairs or from LIDAR flights which are done to create surface elevation models (compare Section 7.3.4). Generally, pseudo orthophotos are easier to generate since displacements of building roofs are not corrected. The tilted views of buildings and other structures of notable height remain unchanged so that the buildings' sides remain partly visible.

10.3. ORTHOPHOTO GENERATION

For accurate measurements, image mosaics and cross-referencing photos to other GIS data, it is necessary to generate orthophotos. The main features of an orthophoto are:

- compensation for the tilt of the aircraft;

- transformation of the central projection of the photo to an orthogonal projection including the correction of elevation-induced scale changes;
- if needed, orientation of the aerial photo towards north (rotation by 90° or 270°), because imaging flights are mostly done in either east-west or west-east direction, to minimize sun illumination effects.

10.3.1 Aerial photo and LOCATIONs preparation

If the aerial photo is available only on an analog medium, the diapositive, it needs to be scanned using a (photogrammetric) backlight scanner. To change a standard flatbed scanner to a backlight scanner a special device is necessary. High quality is important to avoid the introduction of internal image distortions.

The scan resolution during the scanning process determines the effective ground resolution. The required value for the scan software can be calculated from the averaged image scale (see above Equation 10.1). The scanning resolution is usually specified in dpi (dots per inch). The ground resolution R_g in relation to the photo map scale is derived from the scale number M and the scanning resolution R_s (first considered in centimeter) as follows:

$$R_g[cm] = \frac{M}{R_s[lines/cm]} \tag{10.3}$$

For example, we have an aerial photo with the average scale 1:10,000. The scale number is M = 10,000 cm accordingly. The desired spatial resolution on ground R_g for the aerial photo is 40 cm (raster cell width and length). After rearranging the Equation 10.3, the scan resolution R_s which has to be defined for the scanner software is calculated as:

$$\begin{aligned} R_s[lines/cm] &= \frac{M}{R_g[cm]} \\ &= \frac{10000}{40cm} = 250\frac{1}{cm} \end{aligned} \tag{10.4}$$

$$\begin{aligned} R_s[dpi] &= \frac{M}{R_g[cm]} * 2.54\frac{cm}{in} \\ &= 250\frac{1}{cm} * 2.54\frac{cm}{in} = 635dpi \end{aligned} \tag{10.5}$$

In our example, the scan resolution has to be set to 635 dpi to obtain an average ground resolution of 40 cm. It is important to know that a certain pixel resolution does not allow to recognize objects of the same size. An object always needs to be covered by several pixels to become recognizable.

Another decision has to be made about the desired color depth, e.g. whether 256 or more colors should be used. The amount of required disk space is increased accordingly; a standard aerial photo scanned at 1200 dpi with 24 bit color depth will need more than 300 MB per image. Once the scan procedure is done, an image processing software (e.g. `gimp`) should be used to ensure that the bright fiducial marks within the fiducial marks are well visible. A zoomed fiducial mark is shown in Figure 10.3. These marks are crucial for the image-rectification task. This test of visibility can be used to define the lowest acceptable scan resolution. While verifying the fiducial marks, it is useful to zoom-in and denote the parameters from the photo's annotation bar: aircraft altitude, (calibrated) focal length and the timestamp.

Generation of the georeferenced target LOCATION. In the process of orthophoto generation, the scanned and imported aerial photo is ortho-rectified into a georeferenced LOCATION. This LOCATION, containing the reference map and the DEM, has to be created first. The default resolution should be defined in respect to the final target ground resolution of the aerial photo(s). Further details about defining a LOCATION are given in Section 3.2. As mentioned above, it may be necessary to interpolate the DEM to a higher resolution, as shown in Section 5.3.4, in order to minimize unwanted displacements due to DEM resolution problems. Before leaving the projected LOCATION it is recommended to adjust the settings such as resolution and current region to the desired resolution and coordinates to enable GRASS to store the orthophoto accordingly.

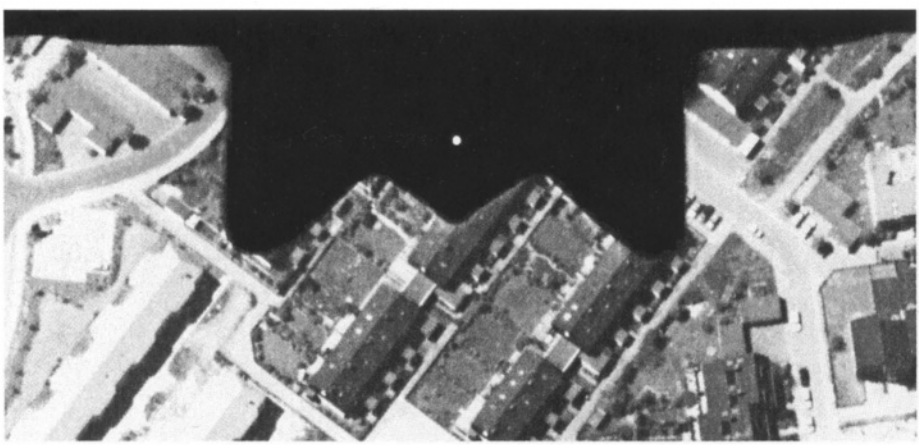


Figure 10.3. Zoomed fiducial mark in an aerial photo. The center of the bright point within the fiducial mark is used during the ortho-rectification process

Creation of the xy LOCATION for the scanned aerial photo. You need to restart GRASS to generate the xy LOCATION for the scanned aerial photo(s). Multiple aerial photos can be imported into the same xy LOCATION. Be sure to define the LOCATION large enough to hold the image(s). When importing photos with `r.in.gdal`, this module optionally extends the current region if needed (flag –e). The resolution is set to one pixel. Then you need to create a separate image group for each aerial photo (see Section 9.3.2). Only in case of multi-channel aerial photos you will add all channels in one group. For our example, the xy LOCATION `imagery` is already available on the GRASS Web site which we will use in this chapter.

10.3.2 Orthophoto generation from vertical aerial photos

The examples provided in this subsection are based on the aerial photos available in the Imagery LOCATION. We will ortho-rectify the photo `gs13.1` into the Spearfish LOCATION and also give additional tips for processing your own data. For our example, the `gs13.1` photo, it is important to have an appropriate high resolution elevation model available in the Spearfish LOCATION. To create it from the existing map `elevation.dem`, you may restart GRASS with this LOCATION, zoom into Spearfish city (located north-west in the Spearfish region) and set the raster resolution to 1m. Now interpolate the elevation model `elevation.dem` to the new resolution within the current region.

Start the generation of orthophoto by restarting GRASS and entering the demo xy LOCATION `imagery` with your name as MAPSET. Then generate an image group `aerial` with `i.group` and select the photo `gs13.1` as member:

```
i.group group=aerial in=gs13.1
```

The GRASS orthophoto module incorporates several photogrammetric tools. After opening a GRASS monitor, it starts with:

```
i.ortho.photo
```

If you have already generated the group `aerial`, it will be automatically loaded. Otherwise, enter the previously created image group containing the aerial photo. Then you reach the main menu:

```
i.ortho.photo --         Imagery Group = aerial

Initialization Options:

   1.      Select/Modify imagery group
   2.      Select/Modify imagery group target
```

```
   3.      Select/Modify target elevation model
   4.      Select/Modify imagery group camera

Transformation Parameter Computations:

   5.      Compute image-to-photo transformation
   6.      Initialize exposure station parameters
   7.      Compute ortho-rectification parameters

Ortho-rectification Option:

   8.      Ortho-rectify imagery files

RETURN   exit

>
```

We will use all menu items sequentially, except for menu item (6) which is required only for oblique aerial images (see Section 10.3.3) or for GPS-equipped flights. It is possible to interrupt the procedure between the steps and continue later; `i.ortho.photo` will store all settings.

1. Select/Modify imagery group. You have already completed this task when starting the module when you had to select a group. Check on top of the menu in the terminal window whether the appropriate image group is selected. If necessary, you can switch to another image group.

2. Select/Modify imagery group target. As a next step, the target LOCATION has to be selected. Enter the name of the projected LOCATION and MAPSET. This menu item corresponds to the module `i.target` (as discussed earlier in Section 9.4.1). For Spearfish enter `spearfish` as TARGET LOCATION and your name as MAPSET (if that MAPSET does not exist, you can use `user1`). To see a list of existing LOCATIONs or MAPSETs enter `list` into the related line.

3. Select/Modify target elevation model. Specify the name of the DEM which is stored in the target LOCATION. For Spearfish enter `elevation.dem` or the recently interpolated elevation model at a higher resolution.

4. Select/Modify imagery group camera. We will now define camera specific parameters. If you previously defined a camera within this MAPSET, you can load it. For our example, we enter the new name `gscamera`. Then the following screen appears:

```
     Please provide the following information:
+-----------------------------------------------------------+
 Camera Name                                 DBA SYSTEMS CAMERA_
 Camera Identification                       _____________________
 Calibrated Focal Length mm.                 0____________________
 Point of Symmetry: X-coordinate mm.         0____________________
 Point of Symmetry: Y-coordinate mm.         0____________________
 Maximum number of fiducial or reseau marks  0____________________
+-----------------------------------------------------------+
```

We enter the following parameters which belong to the photo `gs13.1` (in MAPSET `user1` you can find the camera parameters file `gscam` which belongs to the aerial photos `gs13.1` and `gs14.1`):

```
     Please provide the following information:
+-----------------------------------------------------------+
 Camera Name                                 gscam_____________
 Camera Identification                       gs-vqcy___________
 Calibrated Focal Length mm.                 152.41____________
 Point of Symmetry: X-coordinate mm.         0____________________
 Point of Symmetry: Y-coordinate mm.         0____________________
 Maximum number of fiducial or reseau marks  8____________________
+-----------------------------------------------------------+
```

Important parameters are the *Calibrated Focal Length* which is usually printed on the aerial photo (or delivered along with the aerial photo in a calibration report, for explanation see Section 10.1). The calibrated focal length is unique for each camera. In our example, the "gscam" focal length is 152.41 mm. The coordinates of the *Point of Symmetry* are kept zero since it is the origin for the image coordinates. If the camera data sheet provides different values for the *Point of Symmetry* than the ones provided by the manufacturer, they have to be specified. The number of fiducial marks can be seen on the aerial photo or taken from the photo accompanying camera description file. Four and eight fiducial marks are common. After leaving this screen by <ESC><ENTER> the screen for definition of the distances of the fiducial maps appears.

These distances are referring to the Point of Symmetry. The distance of the bright point inside the fiducial marks with respect to the point of symmetry has to be entered. The fiducial marks may be numbered as desired, but once defined, the numbering sequence must be kept identical to avoid reference errors from assignments of coordinates. If the calibration report defines the order, it should be used. Examples of photos with four or eight fiducial marks are given in Figure 10.4.

In a perfect 23 cm $*$ 23 cm (9" $*$ 9") aerial photo, the distance of the bright points is 113 mm from the point of symmetry. The distances depend on the camera type and will be provided by the camera manufacturer. According to the coordinate system, with its point of origin being set to the image center,

we define the following values for the `gs13.1` aerial photo (all values are in millimeters). You may start with the upper left fiducial mark (no. 1), the other marks are numbered clockwise, until we reach the middle fiducial mark on the left side (no. 8, compare Fig 10.4). The coordinates are entered accordingly:

```
        Please provide the following information:
+-------------------------------------------------------------+
          Fid#     Fid Id          Xf              Yf

           1       1_____      -106.01___      106.01____
           2       2_____      0_________      106.01____
           3       3_____      106.01____      106.01____
           4       4_____      106.01____      0_________
           5       5_____      106.01____      -106.01___
           6       6_____      0_________      -106.01___
           7       7_____      -106.01___      -106.01___
           8       8_____      -106.01___      0_________

                              Next: end__
+-------------------------------------------------------------+
```

Now all required camera data are entered and we can go back to the main menu.

5. Compute image-to-photo transformation. Now we will establish the "interior orientation" of the image, which describes the relation between the physical extent of the aerial diapositive (in millimeters) and its pixels. The coordinate system is centered between the fiducial maps on the photo. The fiducial marks coordinates (as listed in the above table) are assigned to the fiducial marks in the image. This is done graphically in the GRASS monitor

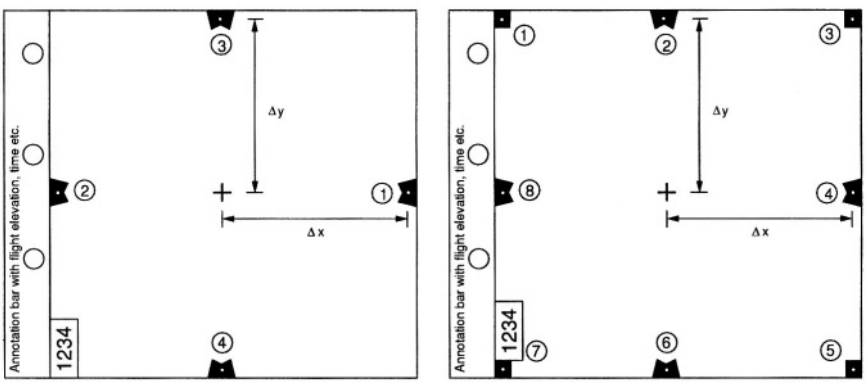


Figure 10.4. Fiducial marks in aerial photo. Left a variant with four fiducial marks, right a variant with 8 fiducial marks. The contents of annotation bar depend on the camera type

by mouse. It is important to refer the table entries carefully to the zoomed centers of the fiducial mark. Small deviations may case large errors in the "exterior orientation" that will be established later.

If the GRASS monitor is not open, you need to exit `i.ortho.photo`, open a monitor, and restart the `i.ortho.photo` module. After selecting this menu entry (no. 5), the aerial photo has to be selected in the monitor. The list of fiducial marks will be automatically displayed. The orthophoto module now expects the assignment of the fiducial coordinate points to the fiducial marks in the image by digitizing.

Using the "zoom" function, enlarge the first mark (mark 1 in the upper left corner of the `gs13.1` photo), until the bright point is visible as composed from multiple pixels. Generally, if no bright point is visible, the scanning resolution was set too low and you will have to start again from scratch by rescanning the aerial photo. In our sample data, the quality of the photo `gs13.1` is unfortunately very low, the fiducial marks in the corners are very difficult to identify. For now you may guess their position – this photo is from 1971, nowadays images are of course of much better quality. The center of the bright point within the enlarged fiducial mark is digitized using mouse by clicking into it. Then the related entry (row) in the fiducial marks table must be selected by double clicking on it. Now both image and fiducial marks coordinates are also shown in the terminal window. The marker in the GRASS monitor will turn to green color once the control point is accepted. This procedure has to be done for all fiducial marks.

The menu item ANALYZE in the monitor allows us to verify the digitizing accuracy. The RMS-error should be less than half pixel size. A misplaced point can be deactivated through a mouse double click on the corresponding control point in the ANALYZE table. This fiducial mark must then be re-digitized again. Once all fiducial marks have been successfully assigned to table entries, select QUIT to go back to the `i.ortho.photo` main menu.

We skip the menu step 6 as it is only needed for rectifying oblique aerial photos as described in Section 10.3.3.

7. Compute ortho-rectification parameters. Now we define the "exterior orientation" which relates the aerial photo to the target coordinate system using ground control points. Reasonable rectification results can be obtained with around twelve control points well distributed over the image. The related elevations, which are crucial for the creation of an orthophoto, are automatically read from the elevation data raster map that we have chosen earlier. To start, select again the aerial photo in the GRASS monitor for display. Then click on the "Plot Cell" entry and into the right part of the monitor to display the reference map. For the example session, you may again select the map `roads`.

Similarly to `i.points`, the corresponding control points are digitized by zooming into a map portion and digitizing the GCPs using mouse both in the aerial photo and in the reference map. When a ground control point pair has been marked in both the aerial photo and the reference map you have to confirm this with a new mouse click (left button: y, right button: n) as explained in the terminal.

Recommended control points are road intersection centers and the centers of objects (buildings etc.). However, be careful when using buildings. An aerial photo will show both the roof and the bottom of a building. Due to the central perspective in the unrectified image only the footprints of buildings may be used, not the displaced roofs. If you want to create a true orthophoto instead of a pseudo orthophoto, the DEM needs to contain the building heights. In this case, the building roofs have to be geocoded, so that both the buildings lower and upper edges are georeferenced.

When more than four GCPs have been digitized, their accuracy can be verified using the ANALYZE menu item. It displays a table of all control points with their RMS error. Each error value is calculated for the control points from the current transformation equations (depending on the camera type, etc). The value is computed for the target LOCATION, so it is given in the projection units of the target LOCATION (usually meters). If a control point appears heavily deviated according to the transformation equations, the row will be shown in red color. Such a control point can be deactivated by double clicking the entry in the ANALYZE table. The total RMS-error is acceptable if it is less than half target resolution of the aerial photo, e.g. below half a meter. Note that the RMS error significantly depends on the digitizing accuracy of the fiducial marks (see above, menu no 5).

If GCPs have been measured by other means, e.g. using a GPS, these coordinates should be entered. Each ground control point is marked within the aerial photo and then "KEYBOARD" is selected as an "input method" instead of the default "SCREEN". When sufficient number (twelve or more GCPs well distributed over the image) of ground control points have been digitized, you can leave the GCPs identification mode with "QUIT" and return to the main menu.

8. Ortho-rectify imagery files. Finally, the ortho-rectification process of the aerial photo can be performed by selecting the menu item (8). Enter a new name for the target orthophoto, in our example we will choose `gs13`. If several aerial photos have been stored in the image group (menu item (1)), they will be all listed here (e.g., multiband aerial photos). Because the defined control points are valid only for the current aerial photo, a new name for the target LOCATION for this photo will be entered here. After leaving this screen, the transformation equations are internally generated ("Computing equations…").

Then another query follows: Selecting "1. Use the current window in the target location" will start the rectification based on the settings in the target LOCATION. Selecting "2. Determine the smallest window which covers the image" allows us to override the settings in the target LOCATION from here, due to the image size and boundaries. The pre-defined values in this screen result from the current aerial photo and the ortho-rectification parameters. Eventually, the ground resolution should be modified to appropriate values and the photo boundaries should be extended to appropriate rounded coordinates.

GRASS will send an email when the rectification process is completed and the new orthophoto generated. Meanwhile, you can exit GRASS and perform other tasks on the computer. Once the email arrives, GRASS can be started with the target LOCATION to view and verify the new orthophoto. An overlay with the reference map will show the quality of the rectification.

10.3.3 Generating orthophotos from oblique aerial photos

The procedure is similar to the one described in Section 10.3.2, but we insert an extra step to take into account the flight and aircraft position parameters, that are needed to generate orthophotos from oblique aerial photos. This step is added after the computation of image-to-photo transformation.

6. Initialize exposure station parameters. In this step, the flight path parameters such as the aircraft tilt and crab, and the camera coordinates and altitude above sea level have to be defined (to be taken from aerial photo auxiliary data). The following three values for the camera position have to be specified (for an example see below):

- X: East aircraft position;
- Y: North aircraft position;
- Z: Flight altitude above sea level

Further the (approximate) tilt of the aircraft has to be specified. This is defined by the angles Omega (roll), Phi (pitch), Kappa (yaw). They represent (see Fig. 10.5, Hildebrandt, 1996:149, Schowengerdt, 1997:95):

- Omega (roll): Raising or lowering of the wings (turning around the aircraft's axis);
- Phi (pitch): Raising or lowering of the aircraft's front (turning around the wings' axis);
- Kappa (yaw): Rotation needed to align the aerial photo to true north: needs to be denoted as +90° for clockwise turn and -90° for a counterclockwise turn.

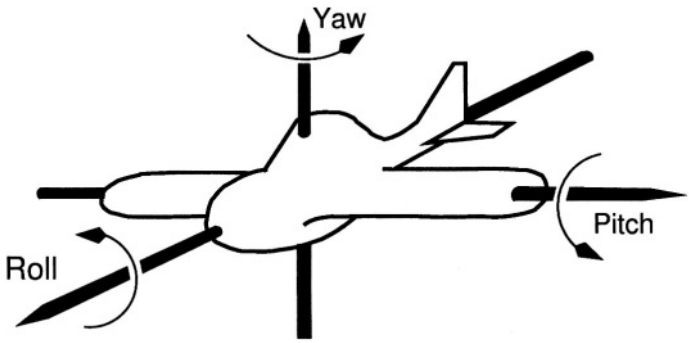


Figure 10.5. Attitude angles of an aircraft

Since Omega and Phi could only be guessed from the aerial photo's annotation bar (if present), approximate angles can be used. Only GPS-supported flights can provide these data accurately.

Then default values for the ortho-rectification calculations have to be entered, to provide initial values for the internal iterative computations. Appropriate value can be derived from the RMS-error. In our example, we set all X, Y and Z to 10m. The question "Use these values at run time? (1=yes, 0=no)" we use "1":

```
        Please provide the following information:
+-------------------------------------------------------------+
 Initial Camera Exposure X-coordinate Meters:     593933__
 Initial Camera Exposure Y-coordinate Meters:     4926053_
 Initial Camera Exposure Z-coordinate Meters:     12192___
 Initial Camera Omega (roll) degrees:             0_______
 Initial Camera Phi (pitch)  degrees:             0_______
 Initial Camera Kappa (yaw)  degrees:             -90_____

 Apriori standard deviation X-coordinate Meters:  10______
 Apriori standard deviation Y-coordinate Meters:  10______
 Apriori standard deviation Z-coordinate Meters:  10______
 Apriori standard deviation Omega (roll) degrees: 0.01____
 Apriori standard deviation Phi (pitch)  degrees: 0.01____
 Apriori standard deviation Kappa (yaw)  degrees: 0.01____

 Use these values at run time? (1=yes, 0=no)      1_
+-------------------------------------------------------------+
```

Now the oblique camera position and additional initial parameters are defined, and we return to the main menu and continue with the step "7. Compute ortho-rectification parameters" and the subsequent procedure as described in the section above.

10.4. SEGMENTATION AND PATTERN RECOGNITION FOR AERIAL IMAGES

Aerial photos can be used for land use/land cover classifications similarly as the satellite data. However, because often only a single channel is available, either in black-and-white, in visible colors or in the infrared spectral range, the reclassification of aerial photos is different from satellite images. Image segmentation is a method for semi-automated feature extraction, such as the land use/land cover classes or edges from remote sensing data. A segmentation algorithm is implemented in the SMAP module `i.smap` which we already introduced in Section 9.8.3 for multi-channel satellite data.

The SMAP algorithm exploits the fact that nearby pixels in an image are likely to belong to the same class. The module segments the image at various scales (resolutions) and uses the course scale segmentations to guide the finer scale segmentations (image pyramid). In addition to reducing the number of misclassifications, the SMAP algorithm generally produces results with larger connected regions of a fixed class which may be useful in numerous applications. The amount of smoothing that is performed in the segmentation is dependent on the behavior of the data in the image. If the data suggest that the nearby pixels often change class, then the algorithm will adaptively reduce the amount of smoothing. This ensures that excessively large regions are not formed. The SMAP segmentation algorithm attempts to improve segmentation accuracy by segmenting the image into regions rather than segmenting each pixel separately.

The size of the submatrix used for segmenting the image has a principle function of controlling memory usage; however, it can also have a subtle effect on the quality of the segmentation in the SMAP mode. The smoothing parameters for the SMAP segmentation are estimated separately for each submatrix. Therefore, if the image has regions with qualitatively different behavior, (e.g., natural woodlands and man-made agricultural fields) it may be useful to use a submatrix small enough so that different smoothing parameters may be used for each distinctive region of the image.

Generating a training map. The module `i.smap` runs with single images as well as multispectral images. The raster polygons resulting from the segmentation process may be vectorized later with `r.poly`. To reclassify, `i.smap` requires a training map containing spectral signature. This training map contains numbered raster polygons which cover selected areas with homogeneous land use/land cover. The pixels inside a training area are considered to be spectrally similar. The training map is analyzed by `i.gensigset` which generates statistical information from the input aerial image based on the

training map. The training map can be digitized (`r.digit` or `v.digit` with `v.to.rast` subsequently). The image statistics derived by `i.gensigset` is input to `i.smap`. As opposed to the Maximum Likelihood Classifier algorithm which operates pixel-wise, the SMAP also considers spectral similarities of adjacent pixels. The module `i.smap` expects the aerial image listed in an image group and subgroup.

If the aerial photo is a color image, it may be analyzed in three channels. During import, a 24 bit aerial color image can be split into the red, green and blue channels. Also black-and-white images can be roughly split into three pseudo-color images with `r.mapcalc` (# operator, see Section 5.2). The # operator can be used to either convert map category values to their grey scale equivalents or to extract red, green, or blue components of a raster map layer into separate raster map layers. The # operator has three forms: `r#map`, `g#map` and `b#map`. These extract the red, green, or blue pseudo-color components in the named raster map, respectively. For example,

```
g.region rast=gs13
r.mapcalc "gs13.r=r#gs13"
r.mapcalc "gs13.g=g#gs13"
r.mapcalc "gs13.b=b#gs13"
d.rgb b=gs13.b g=gs13.g r=gs13.r
```

results in three pseudo-color channels which may be used for the segmentation. However, when working with 24bit images, the information contents is certainly higher.

Sometimes it is useful to generate and additionally use synthetic channels for the classification. For example, the module `r.texture` creates raster maps with textural features from image channels. The textural features are calculated from spatial dependence matrices at 0, 45, 90, and 135 degrees within a given moving window. For details, please refer to the related manual page.

A sample segmentation session. A sample segmentation procedure for the aerial image `gs13` which was previously ortho-rectified into the Spearfish LOCATION may be performed as follows:

```
g.region rast=gs13
i.group group=segment subgroup=segment in=gs13

#digitizing training areas such as fields, forest, roads etc,
#save as map "training":
r.digit
d.rast training

#generate class statistics from training map:
i.gensigset training gr=segment sub=segment sig=smapsig
```

```
#run segmentation:
i.smap gr=segment su=segment sig=smapsig out=gs13.smap
d.rast gs13.smap

#vectorization and visual./report of land use/land cover map:
r.poly -l in=gs13.smap out=gs13.smap
v.support gs13.smap
d.vect gs13.smap
v.report gs13.smap type=area units=h
```

Depending on the training map, the result can be a vectorized land use/land cover map. However, this map may contain a lot of spurious areas. These can be filtered with the script `r.reclass.area`. As an example, only areas greater than 0.25 hectares will be preserved (note, this script only operates in georeferenced LOCATIONS):

```
r.reclass.area -g 0.25 gs13.smap gs13.smap.filt
d.rast gs13.smap.filt
```

The deleted areas are filled with NULL (no-data) values. To reassign values to these NULL cells, we can use a mode filter which replaces the NULL cells by the dominating land use value in the 3x3 neighborhood. The mode filter is implemented in `r.neighbors`:

```
#mode filter to replace NULL cells:
r.neighbors gs13.smap.filt out=gs13.smap2 method=mode size=3
d.rast gs13.smap2

#vectorization:
r.poly -l in=gs13.smap2 out=gs13.smap2
v.support gs13.smap2
d.rast gs13
d.vect gs13.smap2 col=yellow

#area report:
v.report gs13.smap2 type=area units=h
```

We have now the land use/land cover map available as raster and vector map layers. Special care has to be taken for shadows resulting from sun which may either be treated as a special class or reduced by image pre-processing.

Chapter 11

NOTES ON GRASS PROGRAMMING

GRASS provides a unique opportunity to improve and extend GIS capabilities by a new code development. The GNU General Public License (GPL) keeps the code as Free Software, while protecting the rights of the individual authors. Because the source code can be studied, modified and published again, there is an ongoing exchange of knowledge, methods and algorithms between GIS and software engineering experts.

To make the development of GIS tools more efficient, GRASS provides a large GIS library with documented application programming interface (API). The code is portable on numerous architectures and operating systems. The available programming documents and an ongoing reorganization of the code base will enable potential developers to better estimate the workload of adding functionality to GRASS. At the time of writing this book the restructuring is work-in-progress.

An important aspect of the GRASS development is the fact that the developers are advanced GRASS users who improve or extend the existing functionality, based on the needs of their daily GIS use in production.

11.1. GRASS PROGRAMMING ENVIRONMENT

Important communication channel supporting GRASS development is the "GRASS 5 developers mailing list". Here advanced users exchange ideas and discuss problems related to code development or bugfixes. As the source code is managed in CVS (Concurrent Versioning System) the development is free from personal constraints, and the core team members can submit changes at any time. Write access to CVS is granted to those who contribute on a regular basis. The "GRASS Programmer's Manual" is managed in CVS as well

and updated regularly. The access to the full source code, either as a released package, or as a weekly CVS snapshot, or directly extracted from CVS, allows the developers to study the code structure of a full featured GIS. We will describe the general code structure of GRASS 5.3 in greater detail below. Note that the code structure for GRASS 5.7 is different and more standardized. Figure 1.1 at the beginning of the book in Section 1.2 shows the current GRASS Development Model.

11.1.1 GRASS source code

The complete GRASS source code is available on the GRASS Web sites. Those who want to participate in the ongoing source code development of GRASS should learn more about the "GRASS-CVS". This electronic management tool (CVS: *Concurrent Versioning System*[1]) for the source code is used in GRASS development since December 1999. The idea of CVS is to enable the developers to have direct read/write access to the GRASS source code for independent development. CVS supports the centralized management of GRASS development, as the developers work on a single code base with restricted write, but public worldwide read access. You can find more information about this topic on the GRASS Web servers. Another advantage is that the CVS client minimizes data transfer after an initial download: during a subsequent synchronization of the local GRASS source code copy with the centralized CVS server ("cvs update" command) only new code changes are transferred through network.

CVS snapshots which cover the latest development are generated on a weekly basis and made available in a package for download. This is useful for skilled users who do not want to learn CVS but who would like to follow the latest development. However, knowledge about how to compile source code packages is required.

To compile the source code, first download either an officially released source code package, or the "GRASS CVS snapshot", or the latest source code directly from CVS. Read the REQUIREMENTS.html file provided with the source code as well as the INSTALL file to make sure that you have all the necessary libraries installed on your system. After extracting the code (not needed when directly accessing the CVS server), the compilation is done with three steps:

```
./configure <parameters>
make
make install
```

Note that the `configure` script may expect parameters depending on your local installation of required libraries. Please refer to the INSTALL file within the source code for further details and explanations.

11.1.2 Methods of GRASS programming

GRASS 5.3 is written in ANSI C programming language. Additionally, UNIX Shell scripts and a few PERL scripts are implemented. The current graphical user interface `tcltkgrass` is based on Tcl/Tk libraries, the `nviz` visualization tool includes OpenGL function calls. The command line parser is part of the GIS library, but it was extended to print the module descriptions (as shown with `help` parameter) in XML. A simple automatic wxPython GUI builder based on the XML-based GRASS user interface descriptions along with a Document Type Definition (DTD) is implemented. Also a preliminary GRASS/JAVA interface has been implemented for GRASS 4.x and is currently undergoing a major update to current GRASS. However, there are a few more ongoing GRASS/JAVA projects linked to the GRASS Web site, see also Section 2.1.2 for references.

The modular concept of GRASS provides huge potential for development. Two basic levels of programming are supported apart from the standard use. Average users will use "script programming" to simplify repeating processes, while advanced users can extend existing code or develop new modules based on the C-API.

For script programming, there are no general limitations. After setting the specific environment variables as discussed below, more or less any UNIX compliant script language can be used. UNIX shell and PERL scripts are already implemented, also Web-based technologies such as Common Gateway Interface (CGI), Hypertext Preprocessor (PHP) and others are applicable. Writing of customized scripts is supported by the availability of numerous implemented scripts (stored in `$GISBASE/scripts/`) which can be studied and adapted. The advantage of scripts is that they may utilize other UNIX tools such as `awk`, `sed`, or `cut`. If you are familiar with the command line usage of GRASS, it resquires only a small step to start writing your own scripts. To write Web-based applications such as developing a "remotely controlled" GIS which dynamically generates Web map pages will certainly need more time. Besides functionality, issues like software ergonomy, speed, and security will play an important role. In Section 13.4 we will demonstrate a simple example how to publish GRASS raster data with a fast Web mapping server.

We can only provide few introductory notes about GRASS C-programming because this book does not intend to replace a general C-programming tutorial. Our objective is to depict the current source code structure to guide newcomers within the huge code base.

11.1.3 Level of integration

There are different levels of integrating new or external functionality into GRASS. We can distinguish between:

- links to external software:
  - loose coupling: linking other GIS to GRASS via data exchange through shared data formats (e.g. using GRASS in a heterogeneous network such as Linux/Samba/MS-Windows along with proprietary GIS software);
  - tight coupling: direct access to GRASS LOCATION through "libgrass" (e.g. to read maps from a LOCATION directly with UMN/MapServer);
- full integration: modification/extension of GRASS functionality (e.g. writing new code in C, JAVA, UNIX Shell, PERL, etc., such as `gstat` and the GRASS/R interface).

In the following sections we provide several examples of scripts and programs illustrating the possibilities for extending GRASS capabilities. To improve legibility, we have typeset these scripts and programs in a language-sensitive formatting. You can download the larger scripts explained in this chapter from the Internet.[2]

11.2. SCRIPT PROGRAMMING

UNIX Shell and PERL scripts can be used to automate procedures which are repeatedly performed for different maps. The standard GRASS installation provides a set of scripts which, from the user's side, mostly behave like modules programmed in C language. The major drawback is that scripts run slower than compiled implementations. Scripts are stored as ASCII files. Because the GRASS modules can be called within the scripts, complex applications can be developed. When writing such a script for the first time, it is useful to perform the task by running the commands step by step in command line mode. Then the UNIX `history` command allows you to view and save the previously entered commands. Another option is to use the UNIX `script` program which logs a session into a text file.

As a simple example, we write a script `d.rast.region` which first adjusts the current region settings to the map which is given as parameter, and then displays the map:

```
#!/bin/sh
#
# This program is Free Software under the GNU GPL (>=v2).
# Adjust the current region settings to a raster map
# specified as parameter, then display the map
```

```
if test "$GISBASE" = ""; then
 echo "You must be in GRASS to run this program."
 exit
fi

#map name is first parameter:
MAP=$1

#zoom:
g.region rast=$MAP
#erase monitor and display map:
d.erase
d.rast $MAP
```

The first line is mandatory for UNIX Shell scripts. After performing a test to ensure that the script is executed in GRASS environment, the first parameter given on command line is stored into a script-internal variable. GRASS modules are then used to perform the desired task. The map name is stored in the new variable `$MAP`. It is a good programming practice to use easy to understand variable names and to document the functionality step-by-step. It is also important to set the proper file permissions after storing the script into file `d.rast.region`:

```
chmod a+x d.rast.region
```

The script may be stored in the directory `$GISBASE/scripts/` or in a special directory which has to be defined by the environmental variable `$GRASS_ADDON_PATH` before starting GRASS itself. This allows you to keep your scripts and modules physically separated from the standard GRASS. The path(s) defined in `$GRASS_ADDON_PATH` are added to the modules search path during startup.

The next example, demonstrating the potential of script programming, slightly extends the GRASS functionality by introduction of the `awk` tool. This script calculates general geostatistical information for raster images (adapted from Albrecht, 1992). You may save this script as `statistics.sh` and set the UNIX execute permissions:

```
#!/bin/sh
# This program is Free Software under the GNU GPL (>=v2).
# calculate univariate statistics for GRASS raster data

if test "$GISBASE" = ""; then
 echo "You must be in GRASS to run this program."
 exit
fi
```

```
MAP=$1
r.stats -1 $MAP | awk 'BEGIN {sum = 0.0 ; sum2 = 0.0}
NR == 1{min = $2 ; max = $1}
      {sum += $1 ; sum2 += $1 * $1 ; N++}
      {
       if ($1 > max) {max = $1}
       if ($1 < min) {min = $1}
      }
END{
print "Number of raster data samples N =",N
print "Minimum value MIN =",min
print "Maximum value MAX =",max
print "Variation     v   =",(max - ((min * -1) * -1))
print "Mean          MEAN =",sum / N
print "Variance      S2   =",(sum2 - sum * sum / N) / N
print "Standard deviation S =",sqrt((sum2 - sum * sum/N) / N)
print "Variation coeffic. V =",100*(sqrt((sum2 - sum*sum/N)/N)) \
      /(sum/N)
}'
```

This script must be used from inside GRASS. The name of the input raster map is specified as a parameter. Within this script the output of the GRASS module `r.stats` is piped to the program `awk`. The statistical calculations are done within `awk`, and the results are printed out. A modified version of the above script, `r.univar`, is provided with GRASS.

The next script example calculates the center of gravity of an area (area centroid). You can use it to find the center of gravity of a region defined by a watershed boundary. The script requires a watershed map generated by `r.watershed`. Internally, all basins except the watershed of interest are masked out with `r.mask`. You can save the following script as a text file `r.centroid`:

```
#!/bin/sh
#
# This program is Free Software under the GNU GPL (>=v2).
# Calculate centroid of raster area (center of gravity)
#
#Parser definitions:
#%Module
#% description: Calculates centroid of raster area (center of gravity)
#%End
#%option
#% key: map
#% type: string
#% gisprompt: old,cell,raster
#% description: raster input map
```

```
#% required: yes
#%end
#%option
#% key: areanumber
#% type: integer
#% description: number of area for centroid calculation
#% required: yes
#%end

if test "$GISBASE" = ""; then
  echo "You must be in GRASS to run this program."
  exit 1
fi

eval 'g.gisenv'
: ${GISBASE?} ${GISDBASE?} ${LOCATION_NAME?} ${MAPSET?}
LOCATION=$GISDBASE/$LOCATION_NAME/$MAPSET

if [ "$1" != "@ARGS_PARSED@" ] ; then
  exec $GISBASE/etc/bin/cmd/g.parser "$0" "$@"
fi

MAP="$GIS_OPT_map"
AREANO=$GIS_OPT_areanumber

# here we go for centroid calculation:
# centroid is defined as
#                N
#  x_c = 1/A * SUM (x_i * a_i)
#               i=1
#
#                M
#  y_c = 1/A * SUM (y_i * a_i)
#               i=1
# with
#  N: total number of cells in x direction
#  M: total number of cells in y direction
#  x_i: distance of cell center from left boundary
#  y_i: distance of cell center from upper boundary
#  a_i: area of ith cell

# calculate area in square meters:
AREA=`r.report -qhen $MAP u=me | sed -e 's/ //' |\
      grep "^|$AREANO|." |\
      cut -d'|' -f4| awk '{printf "%.2f", $1}'`

#is area not present in map?
if [ ! $AREA ] ; then
  echo "ERROR: Selected area $AREANO not found in map $MAP."
```

```
  exit 1
fi

# determine current resolution and LOCATION units:
EWRES='awk ' /e-w/ { print $3}' $LOCATION/WIND'
NSRES='awk ' /n-s/ { print $3}' $LOCATION/WIND'
if [ -f $LOCATION/../PERMANENT/PROJ_UNITS ] ; then
  UNITS='cat $LOCATION/../PERMANENT/PROJ_UNITS |\
        awk '/units:/ {print $2}''
else
  UNITS="cellunits"
fi

echo "Area of basin $AREANO: $AREA meters^2"
echo "Current cell resol. [$UNITS]: EW: $EWRES, NS: $NSRES"

#set MASK to get only selected area:
g.rename rast=MASK,$TMP.MASK 2> /dev/null
r.mapcalc MASK="if($MAP == $AREANO)"
echo "Showing selected area..."
d.rast $MAP

echo "Calculating x_min and x_min of area..."
#calculate x_min
XMIN='r.stats -1gnq $MAP |cut -d ' ' -f1 | awk 'BEGIN{min = 0.0}
NR == 1{min = $1}
      {if ($1 < min) {min = $1}}
END{print min}''

#calculate y_min
YMIN='r.stats -1gnq $MAP |cut -d ' ' -f2 | awk 'BEGIN{min = 0.0}
NR == 1{min = $1}
      {if ($1 < min) {min = $1}}
END{print min}''

echo "Calculating centroid..."
# calculate x_c:
r.stats -1gnq $MAP |cut -d' ' -f1 | awk 'BEGIN{
    sum = 0.0 ; calc = 0.0 ; xmin2 = 0.0
    ewres = '$EWRES' ; nsres = '$NSRES'
    xmin = '$XMIN' ; area = '$AREA'}
 NR == 1{xmin2 = xmin * 1.0}
       {calc = ($1 - xmin2) * ewres * nsres}
       {sum = sum + calc}
 END{printf "Center of gravity x_c: %.2f\n", sum/area + xmin2}'

# calculate y_c:
r.stats -1gnq $MAP |cut -d' ' -f2 | awk 'BEGIN{
    sum = 0.0 ; calc = 0.0 ; ymin2 = 0.0
```

```
   ewres = '$EWRES' ; nsres = '$NSRES'
   ymin = '$YMIN' ; area = '$AREA'}
NR == 1{ymin2 = ymin * 1.0 }
      {calc = ($1 - ymin2) * ewres * nsres}
      {sum = sum + calc}
END{printf "Center of gravity y_c: %.2f\n", sum/area+ymin2}'

#restore old MASK, if present
g.remove MASK > /dev/null
g.rename rast=$TMP.MASK,MASK 2> /dev/null
```

This script shows how UNIX commands and GRASS modules can be linked. It also provides a simple parser (through `g.parser`), so the script will both run both in the command line mode, and, when started without parameters, in interactive mode. The first test checks if the module is started within GRASS, and exits if it is not. The usage description is stored in a function to save space within the script and to improve the legibility.

With `g.parser` a simple parser is provided to query input maps. Here a raster map such as a watershed or other raster area map are of interest. The module checks if the user requested the module help description or otherwise accepts the first parameter as raster file name. Then the ID number of the raster polygon is checked (which might be queried with `d.what.rast` earlier). This is followed by the centroid calculation according to the centroid formula. The calculation requires the area of the current raster polygon which is retrieved from `r.report`. Additionally the current resolution is needed. In case that MASK is present, it will be saved and later restored. This is necessary, as the area of interest has to be selected inside the input raster map with a new MASK. To see the `r.centroid` script operating on Spearfish region data, you may run:

```
#calculate watershed (minimum area size: 1000 cell units):
g.region rast=elevation.dem -p
r.watershed elevation=elevation.dem basin=basins threshold=1000
d.rast basins

#calculate area and centroid coordinates for watershed no. 42:
r.centroid map=basins areanumber=42

# display centroid of watershed no. 42,
# coordinates from above r.centroid:
echo "593934.29 4918883.74 42" | s.in.ascii sites=grav_center
d.sites grav_center col=red
```

Generally you may face the problem that a script is not working as expected. To identify problem(s), you can add printing of variable contents

with `echo $VARIABLE`. An alternative, recommended method to debug shell scripts is to start them with `-x` flag. This will switch the shell into an echo-mode:

```
sh -x r.centroid map=basins areanumber=42
```

This will echo every line in the terminal window which simplifies error identification.

Further scripts are described in Albrecht, 1992 and Shapiro and Westervelt, 1992. Plenty of scripts to learn from are included in GRASS (from inside GRASS change to `$GISBASE/scripts/`).

11.3. AUTOMATED USAGE OF GRASS

Due to the modular character of GRASS a monolithic "GRASS program" does not exist. In fact GRASS is a collection of modules which are run in a special environment. The structure allows GRASS to be completely controlled from outside through scripts.

GRASS in batch mode. The usage of GRASS in batch mode requires setting of some environment variables, which can be also done manually or in scripts. An example in "bash-shell" style may be as follows:

```
echo "LOCATION_NAME: spearfish"                > $HOME/.grassrc5
echo "MAPSET: user1"                           >> $HOME/.grassrc5
echo "DIGITIZER: none"                         >> $HOME/.grassrc5
echo "GISDBASE: /usr/local/share/grassdata" >> $HOME/.grassrc5
echo "GRASS_GUI: text"                         >> $HOME/.grassrc5

export GISBASE=/usr/local/grass53
export GISRC=$HOME/.grassrc5
export PATH=$PATH:$GISBASE/bin:$GISBASE/scripts
```

After setting these variables, the environment is defined and all GRASS modules can be used. Note that GRASS 5.3 allows the user to run only a single session at a time. Otherwise region setting conflicts may arise. However, the user can work in multiple session in the same LOCATION, when using different MAPSETs. The module-specific environment variables are further explained in the software documentation, check with:

```
g.manual env_vars
```

Once the minimum number of environment variables has been set, GRASS commands can be integrated into shell, CGI, PERL, PHP and other scripts. A CGI-based example to build a mapping server on top of GRASS is "SlideLinks"[3] on Internet which also integrates PHP and a PostgreSQL database management system for landslide inventory. From time to time it is recommended to remove temporary GRASS files by:

```
$GISBASE/etc/clean_temp
```

Automated generating of a LOCATION from external raster data. A nice application for running GRASS in batch mode is the automated generating of LOCATIONs from external GIS raster data. We use `r.in.gdal` for this purpose as the module supports many formats and it can also read projections from metadata if provided:

```
#!/bin/sh
# This program is Free Software under the GNU GPL (>=v2).
# create a new LOCATION from a raster data set

#variables to customize:
GISBASE=/usr/local/grass53
GISDBASE=/usr/local/share/grassdata
MAP=$1
LOCATION=$2

#nothing to change below:
if [ $# -lt 2 ] ; then
 echo "Script to create a new LOCATION from a raster data set"
 echo "Usage:"
 echo "   create_location.sh rasterfile location_name"
 exit 1
fi

#generate temporal LOCATION:
TEMPDIR=$$.tmp
mkdir -p $GISDBASE/$TEMPDIR/temp

#save existing $HOME/.grassrc5
if test -e $HOME/.grassrc5 ; then
   mv $HOME/.grassrc5 /tmp/$TEMPDIR.grassrc5
fi

echo "LOCATION_NAME: $TEMPDIR" > $HOME/.grassrc5
echo "MAPSET: temp"            >> $HOME/.grassrc5
echo "DIGITIZER: none"         >> $HOME/.grassrc5
echo "GISDBASE: $GISDBASE"     >> $HOME/.grassrc5
export GISBASE=$GISBASE
export GISRC=$HOME/.grassrc5
export PATH=$PATH:$GISBASE/bin:$GISBASE/scripts
```

```
# import raster map into new location:
r.in.gdal -oe in=$MAP out=$MAP location=$LOCATION
if [ $? -eq 1 ] ; then
  echo "An error occured. Stop."
  exit 1
fi

#restore saved $HOME/.grassrc5
if test -f /tmp/$TEMPDIR.grassrc5 ; then
   mv /tmp/$TEMPDIR.grassrc5 $HOME/.grassrc5
fi

echo "Now launch GRASS with:"
echo " grass53 $GISDBASE/$LOCATION/PERMANENT"
```

In case that no projection information is found in the raster data set, you may use `g.setproj` to later store the projection information. Note that `g.setproj` does not reproject any data. The above script accepts only raster data sets. Because a general import tool for vector data does not yet exist in GRASS 5.3 (but in GRASS 5.7), the script for the same task based on vector data would require vector import modules and a format detector based on file extensions. Another approach would be to query the data format as additional third parameter. In this way the script can be extended into a general LOCATION creation and import script.

Another method to launch GRASS directly without manually specifying the names of LOCATION, MAPSET and DATABASE is:

```
grass53 /usr/local/share/grassdata/spearfish/user1
```

This is only successful if the LOCATION and MAPSET exist.

11.4. NOTES ON PROGRAMMING GRASS MODULES IN C

This section explains the GRASS code organization for a user with basic C language programming knowledge. While we cannot explain GRASS programming within this book, we provide a brief introduction to the huge code base. GRASS provides an ANSI C language API with several hundred GIS functions, from reading and writing maps to area and distance calculations for georeferenced data as well as attribute handling and map visualization. All important aspects of GRASS programming are covered in the "GRASS 5.3 Programmer's Manual" (available from the GRASS Web site). To understand the usage of the GRASS API it is helpful to explore the existing modules. The

general structure of modules is similar, with each module stored in a directory of the GRASS source code. You can find further useful tips for GRASS programming style as well as recommendations to avoid portability traps in the file `SUBMITTING`. This file is included in the source code.

It is important to know that the GRASS modules are linked against an internal "front.end". The "front.end" module will call the interactive version of the command if there are no command-line arguments entered by the user. Otherwise, it will run the command-line version. If only one version of the specific command exists (for example, if there is only a command-line version available) the existing command is executed. Code parameters and flags are defined within each module. They are used to ask user to define map names and other options.

GRASS source code structure. The GRASS 5.3 source code structure is as follows:

- GRASS GIS library (only the most relevant components are listed):

```
- src/CMD/              # internal scripts for compilation
- src/include/          # header files
- src/libes/            # GIS library routines
- src/display/devices   # display and file drivers
- src/fonts/            # character fonts
- src/front.end/        # internal routines for the
                              interactive mode of modules
```

- Modules (standard tree):

```
- src/display/          # modules for displaying maps in the
                              GRASS monitor
- src/general/          # file management modules
- src/imagery/          # image processing modules
- src/mapdev/           # vector modules
- src/misc/             # miscellaneous modules
- src/paint/            # paint driver (PPM)
- src/ps.map/           # postscript driver
- src/raster/           # raster modules
- src/scripts/          # scripts
- src/sites/            # sites modules
- src/tcltkgrass/       # Tcl/Tk GUI
- html/                 # modules descriptions
```

- Contributions form various institutions (additional contributions are in the standard tree):

```
- src.contrib/
```

- Modules with linked simulation models and various interfaces:

```
- src.garden/
```

Note that this structure is subject to change in future releases.

The "GRASS programming library" (C API) is structured as follows (typical function name prefixes for related library functions are listed in squared brackets):

- GIS library: database routines (GRASS file management), memory management, parser (parameter identification on command line), projections, raster data management etc. [G_]
- vector library: management of area, line, and point vector data [Vect_, V2_, dig_]
- image data library: image processing file management [I_]
- site data library: site data management [G_sites_]
- display library: graphical output to the monitor [D_]
- raster graphics library: display raster graphics on devices [R_]
- segment library: segmented data management [segment_]
- vask library: control of cursor keys etc. [V_]
- rowio library: for parallel row analysis of raster data [rowio_]

Modules consist of one or more C program files (*.c), the local header files (*.h) and a Gmakefile. GRASS 5.3 is provided with its own "make" routine: gmake53. The file Gmakefile contains instructions about files to be compiled and libraries to be used (GRASS and UNIX libraries). The GRASS libraries are predefined as variables. The structure of Gmakefile follows special rules. A simple example illustrates a typical Gmakefile (it is important to generate indents with <TAB>, not with blanks!):

```
PGM=i.sat.reflectance
HOME=$(BIN_CMD)

LIBES= $(IMAGERYLIB) $(GISLIB) $(VASKLIB) $(VASK)
DEPLIBS=$(DEPIMAGERYLIB) $(DEPGISLIB)
OBJ = main.o\
        open.o\
        atmos.o\
        sun_pos.o\
        correction.o\
        histogram.o\
        history.o
```

```
$(HOME)/$(PGM): $(OBJ) $(LIBES)
        $(CC) $(LDFLAGS) -o $@ $(LIBES) $(MATHLIB) $(XDRLIB)

$(LIST): global.h

$(IMAGERYLIB): #
$(GISLIB): #
$(VASKLIB): #
```

The line `$(HOME)/$(PGM) ...` and the following line contain compiler instructions. Above this line several variables are set which contain library names, the name of this GRASS module and the target directory for storing the compiled code. Numerous variables used here are pre-defined in:

```
src/CMD/head/head.$ARCH
```

The variable `$ARCH` is extended by the name of the system's architecture on which GRASS is compiled (e.g. `head.i686-pc-linux-gnu`). This "head" file is created depending on the platform by `configure` script, which has to be run before a first compilation of GRASS. It contains information related to the compiler, paths to local libraries and include files etc. The other internal variables are defined in

```
src/CMD/generic/make.mid
```

These settings should be kept unchanged.

It is a good programming practice to subdivide a C program (here a GRASS module) into several files, organized by functionality. All these files have to be listed in the objects list in the `Gmakefile`. GRASS GIS library commands can be used in the source code when the code is linked against the related libraries. A short example of a raster module (file `main.c`):

```
/*
 * This program is Free Software under the GNU GPL (>=v2).
 * Conversion of LANDSAT-TM5 DNs to at-sensor radiances
 */

#include <stdio.h>
#include <string.h>
#include <math.h>
#include "gis.h"

int main(int argc, char *argv[])
{
    CELL *cellbuf;
    DCELL *result_cell;
```

```
int nrows, ncols;
int row, col;
char *groupname;
int fd;
struct GModule *module;
struct
{
    struct Option *group;
} parm;
struct
{
    struct Flag *quiet;
} flag;

module = G_define_module();
module->description = "Module to convert LANDSAT-TM5 "
    "digital numbers to at-sensor radiances";

parm.group = G_define_option();
parm.group->key = "group";
parm.group->type = TYPE_STRING;
parm.group->required = YES;
parm.group->description = "Imagery group of images to\
                          be converted";

flag.quiet = G_define_flag();
flag.quiet->key = 'q';
flag.quiet->description = "Run quietly";

G_gisinit (argv[0]);

if (G_parser(argc,argv))
    exit(-1);
groupname = parm.group->answer;

/* function defined in other file */
open_file(groupname, fd);

nrows = G_window_rows();
ncols = G_window_cols();
cellbuf = G_allocate_raster_buf(CELL_TYPE);
result_cell = G_allocate_raster_buf(DCELL_TYPE);

/* go row wise and col wise through image */
for (row = 0; row < nrows; row++)          /* rows loop */
{
      /* read integer satellite channel */
      G_get_raster_row (fd, cellbuf, row, CELL_TYPE);
      for (col = 0; col < ncols; col++)    /* cols loop */
```

```
            {
                /* the formula is defined in another file: */
                result_cell[col] = calc_new_pixel(cellbuf);
            }  /* end cols loop */

            G_put_raster_row(fd, result_cell, DCELL_TYPE);
    } /* end rows loop */

    G_close_cell (fd);
    return 0;
}
```

The calculation is done row-wise and column-wise (see "for" loop). This draft program illustrates only the general structure of GRASS code, for copyright reasons it is not a real GRASS program. Please refer to the GRASS source code for the real world implementations.

Future of GRASS programming. At the time of writing this book an initial GRASS 5.7 version providing the new 2D/3D vector engine was published. The developments for GRASS 5.3 will be shifted to GRASS 5.7. Migration of the code to the new code repository will be accompanied by a major code cleanup. Besides a more standardized code structure the code spread in various modules will be organized into new library functions. Existing library functions will be examined for consistency, and if needed, functions performing similar tasks will be merged. Also, at the module level, merging of modules with similar functionality will be done. In general the goal is to provide a well defined, layered GRASS model with GRASS-Core, providing all library functions, GRASS-Base, providing basic modules for importing, exporting, displaying and basic manipulation of spatial data sets and extended GRASS packages including specialized add-on packages for image processing, hydrologic modeling, volume data management and analysis, etc.

GRASS intends to be a general purpose GIS. The new GRASS 5.7 version is a major step to develop a reliable, intuitive to use, flexible GIS in terms of Free Software. Skilled users are invited to participate in it's further development.

NOTES

1 CVS software, `http://www.cvshome.org`
2 GRASS Tutorials Web site,
`http://mpa.itc.it/grasstutor/`
3 SlideLinks – GRASS Web server,
`http://gisws.media.osaka-cu.ac.jp/slidelinks/`

Chapter 12

USING GRASS: APPLICATION EXAMPLES

In this chapter, we explain the practical use of GRASS using examples in the area of natural resources. First, we illustrate simple modeling using the Spearfish data set. More complex application is included in the section focused on land use management. To save space, we use the command line mode throughout this chapter; however, the same tasks can be performed by accessing the commands using the TclTk interface or an interactive mode (see Section 3.1.3).

12.1. WORKING WITH DIGITAL ELEVATION MODELS: EROSION RISK ASSESSMENT

In this section, we will use topographic analysis and map algebra to find locations with high erosion rates. To compute simplified erosion risk maps, we can use the widely accepted Universal Soil Loss Equation (USLE), modified for complex terrain (see Mitasova et al., 2001). It is often referred to as USLE3D or the updated RUSLE3D. The general formulation for both USLE and RUSLE is defined as follows:

$$A = RKLSCP \tag{12.1}$$

where A is average annual soil loss in ton/(acre.year)=0.2242kg/(m^2.year), R is rainfall factor in (hundreds of ft-tonf.in)/(acre.hr.year)=17.02 (MJ.mm)/(ha.hr.year), K is soil erodibility factor in (ton acre.hr)/(hundreds of acre ft-tonf.in)=0.1317(ton.ha.hr)/(ha.MJ.mm), LS is a dimensionless topographic (length-slope) factor, C is a dimensionless land cover factor, and P is a dimensionless prevention measures factor. The major difference between USLE and RUSLE is in the values of these factors and the methodology for

their derivation (you can download the field-based RUSLE and its documentation from Internet[1]). USLE was originally designed for uniform planar fields with slope steepness and slope length estimated in the field. It has been shown that the standard form of the *LS* factor can be modified to better express the influence of complex terrain (Moore and Burch, 1986, Mitasova et al., 1996, Desmet and Govers, 1996, Moore and Wilson, 1992). The modified factor, representing topographic potential for erosion at a point on the hillslope, is a function of the upslope area per unit width and the slope angle:

$$LS = (m+1)\left(\frac{U}{22.1}\right)^{m}\left(\frac{\sin\beta}{0.09}\right)^{n} \qquad (12.2)$$

where U is the upslope area per unit width (measure of water flow) in meters (m^2/m), β is the slope angle in degree, 22.1 is the length of the standard USLE plot in meters and 0.09 = 9% = 5.15° is the slope of the standard USLE plot. The values of exponents range for m = 0.2 – 0.6 and n = 1.0 – 1.3, where the lower values are used for prevailing sheet flow and higher values for prevailing rill flow. When nothing is known about the type of flow, m = 0.4 and n = 1.3 are usually used (see RUSLE for ArcView[2]). We will derive the required parameters using the data available in our Spearfish data set.

12.1.1 Computation of the LS factor

We start GRASS typing `grass53` and select `spearfish` and `user1` for LOCATION and MAPSET. By running `g.list rast` we can see that we have some elevation data, as well as the soil and vegetation map layers that we need for estimation of erosion risk.

First, we compute the *LS* factor using the older 30 m resolution DEM `elevation.dem`, as both upslope area and slope are derived from a DEM. To make sure that our data are not inappropriately resampled, we set the region to the one defined by the elevation map layer. We can then compute the flow accumulation and slope as follows:

```
g.region -p rast=elevation.dem
r.flow elevation.dem dsout=flowacc.30m
r.slope.aspect elevation.dem slope=slope.30m aspect=aspect.30m
d.rast flowacc.30m
d.rast slope.30m
```

Note that `r.flow` creates a special non-linear color table for the flow accumulation map layer so that the spatial pattern of flow is visible both on the hillslopes, where the values are very low, and in the valleys, where flow accumulation can be several magnitudes higher.

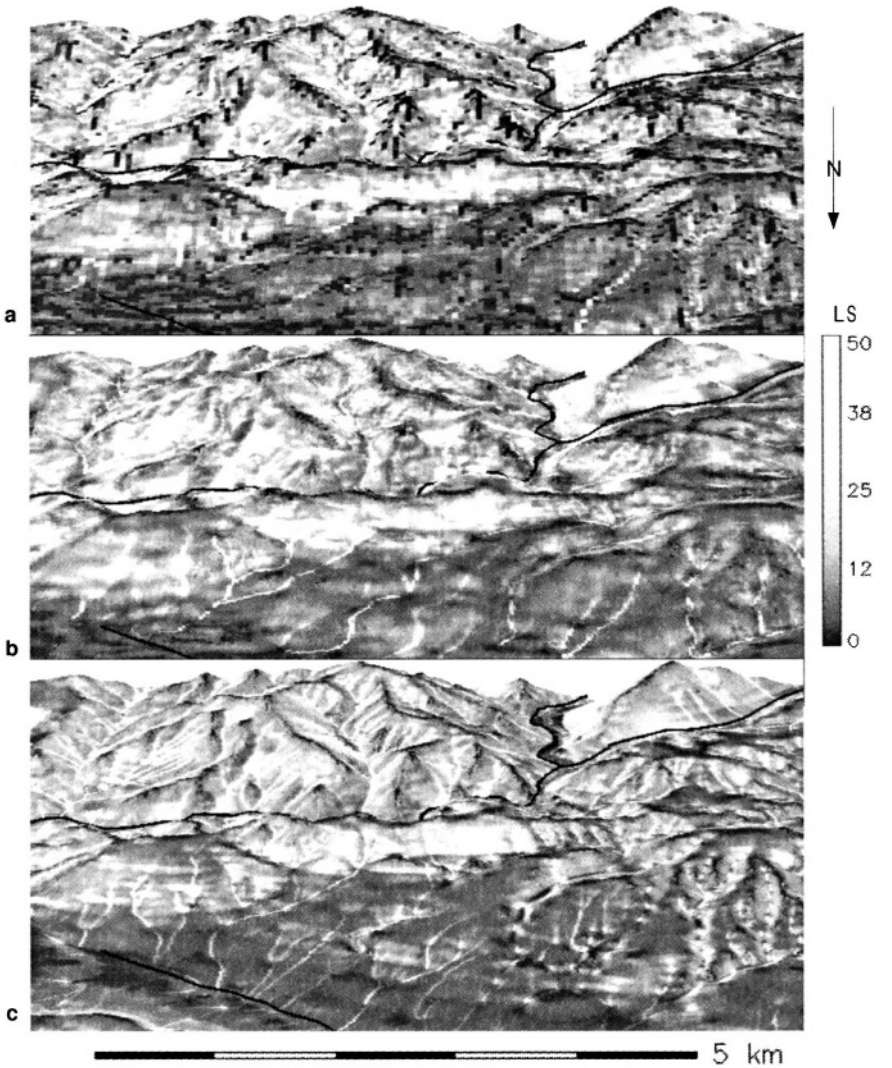


Figure 12.1. LS factor computed at a) 30 m resolution using the original Speafish integer DEM, b) 15 m resolution using the re-interpolated floating point DEM, c) 10 m resolution using the new USGS DEM from National Elevation Dataset. The center of this zoomed-in area is located at -103.68632, 44.42721 longitude, latitude. White areas have high topographic potential for erosion, dark grey areas are stable

When computing the *LS* factor, we multiply the flow accumulation by the resolution to get the upslope contributing area per contour width (it is in fact the flowline density multiplied by cell area in m^2 divided by cell width in m) and calculate the Equation 12.2 using `r.mapcalc`:

```
r.mapcalc "lsfactor.30m=1.4*exp(flowacc.30m*30./22.1,0.4)\
                          *exp(sin(slope.30m)/0.09,1.3)"
```

Before displaying the `lsfactor.30m` map layer, we assign it a special color table to account for its skewed distribution (similarly to flow accumulation):

```
r.colors lsfactor.30m col=rules
 fp: Data range is 0.000 to 182.750
> 0 200 255 200
> 5 yellow
> 10 orange
> 20 red
> 50 magenta
> 400 black
> end

d.rast lsfactor.30m
```

Because this is a floating point map representing a continuous field, the colors smoothly change between the defined color values. The first color is light green, defined as RGB (see Chapter 8). The last black color is defined for a value higher than the maximum *LS* factor, so that the map color for the highest *LS* factor is darker magenta. It also makes the color table usable for map layers with a greater range of values than we have in `lsfactor.30m` that we will compute later. When we display the map, we can see that we have a substantial area with zero topographic factor for erosion (Figure 12.1). This indicates that the vertical precision the DEM (given as integer values in meters) may not be adequate.

Re-interpolation of DEM. We will try to improve the representation of topography by re-interpolating the DEM from 30 m resolution integer z-values to 15 m resolution floating point z-values. While the re-interpolated DEM does not include more information than the original elevation data, its digital representation is improved by removal of the flat areas and "steps" due to the integer representation, by better description of "curved" terrain features such as valleys and shoulders through a finer grid. We will use the RST method (see Section 7.3.2, Mitas and Mitasova, 1999) for re-interpolation. Note that this step is not necessary if the DEM already has adequate resolution and precision.

We will base the re-interpolation on random points generated from the original integer DEM. First, make sure that the region (extent and resolution) is the same as the original elevation map by running `g.region`. Then, sample the existing DEM with 100,000 random points (you can use `r.info` to find the number of cells in your DEM and then choose about 1 point for 2-3 cells):

```
g.region rast=elevation.dem -p
r.info elevation.dem
r.random elevation.dem nsites=100000 sites_out=elev30.sites
```

Next, change the resolution to 15 m and interpolate the newly created sites map layer using `s.surf.rst` with lowered tension and increased smoothing parameter to reduce the impact of "steps" and noise in the low elevation areas (see Section 7.3.2 for more details on parameters for spline interpolation). Because we will also need slope angle, we add the computation of topographic parameters to the interpolation. Finally, we add the UNIX & symbol to make it run in the background, because interpolating a 932 x 1266 grid will take some time:

```
g.region -p res=15
s.surf.rst in=elev30.sites elev=elev.15m ten=30 smooth=1.0\
                          slope=slope.15m aspect=aspect.15m &
```

We can now use this re-interpolated and smoothed DEM to derive a new *LS* factor:

```
r.flow elev.15m dsout=flowacc.15m
r.mapcalc "lsfactor.15m=1.4*exp(flowacc.15m*15./22.1,0.4)\
                             *exp(sin(slope.15m)/0.09,1.3)"
r.colors lsfactor.15m rast=lsfactor.30m
d.erase
d.rast lsfactor.15m
```

To make the visual comparison with the `lsfac.30m` easier, we have assigned the new `lsfactor.15m` the same color table.

Recently, a new 10 m resolution DEM has become available for the Spearfish area; it is included in the data set as `elevation.10m`. You can set the region to the new DEM and compute the refined *LS* factor as follows:

```
g.region -p rast=elevation.10m
r.flow elevation.10m dsout=flowacc.10m
r.slope.aspect elevation.10m slope=slope.10m aspect=aspect.10m
r.mapcalc "lsfactor.10m=1.4*exp(flowacc.10m*10./22.1,0.4) \
                               *exp(sin(slope.10m)/0.09,1.3)"
r.colors lsfactor.10m rast=lsfactor.30m
d.erase
d.rast lsfactor.10m
nviz elevation.10m col=lsfactor.10m
```

You can see much more detail in the resulting map, especially when displayed in 3D using `nviz` (Figure 12.1). Noise has been greatly reduced, but there are some waves along contours and local pits, typical for spline interpolation with tension set too high (see Section 7.3). Unfortunately, USGS does not provide details about the method used to create this new product.

Comparison of the 10 m, 15 m and 30 m resolution *LS* factors. The visual comparison of the `lsfactor.10m`, `lsfactor.15m`, and `lsfactor.30m` maps indicates that there is a larger extent of areas where $LS = 0.0$ in the 30 m

resolution map than in the 15 m and 10 m resolution result (Figure 12.1). You can run

```
d.rast lsfactor.30m val=0
d.rast lsfactor.15m val=0
d.rast lsfactor.10m val=0
```

to see the difference. To quantify it, we can compare the summary statistics for the results by running `r.univar`. To make sure that the comparison is correct, we apply MASK so that the NULLs present in `lsfactor.30m` (due to the NULLs in the original DEM) are also counted as NULLs in `lsfactor.15m` and `lsfactor.10m`, (see Section 5.1.7 on how to apply a MASK):

```
g.region -p rast=lsfactor.30m
r.mapcalc "maskdem=if(elevation.dem,1)"
g.copy rast=maskdem,MASK
r.univar lsfactor.30m
[...]
 Number of cells (excluding NULL cells): 290123
 Minimum: 0
 Maximum: 182.7502010664
 Range:  182.75
 Arithmetic mean:  6.90423
 Variance:  63.3764
 Standard deviation: 7.96093
 Variation coefficient: 115.306 %

g.region -p rast=lsfactor.15m
r.univar lsfactor.15m
[...]
 Number of cells (excluding NULL cells): 1169268
 Minimum: 0
 Maximum: 279.8847366021
 Range: 279.885
 Arithmetic mean: 7.35935
 Variance: 61.6206
 Standard deviation: 7.84988
 Variation coefficient: 106.665 %

g.region -p rast=lsfactor.10m
r.univar lsfactor.10m
[...]
 Number of cells (excluding NULL cells): 2626610
 Minimum: 0
 Maximum: 274.7339156185
 Range: 274.734
 Arithmetic mean: 7.77852
 Variance: 75.6691
 Standard deviation: 8.6988
 Variation coefficient: 111.831 %
```

Note that you may get slightly different numbers for `lsfactor.15m` because your DEM will be re-computed from a different set of random points than the one used in this book. The results show that the mean *LS* factor is lower for the 30 m resolution, which was expected from the visual comparison. The reinterpolation of the DEM to 15 m resolution brings the result closer to the values derived from the new, more accurate 10 m resolution DEM. The difference in range is much larger; however, displaying the cells with values over 180 by `d.rast lsfactor.15m val=180-280` reveals that the higher values are only present in very few cells (41 out of over 1 million). These are mostly due to the fact that smoothing removed some of the smaller pits and the flowlines are generally longer. To quantify the visible differences in the extent of *LS* = 0.0 values and assess their impact, we extract them from the `lsfactor.30m`, `lsfactor.15m`, and `lsfactor.10m` map layers using `r.mapcalc` and compute their proportion with `r.report`:

```
g.region rast=lsfactor.30m
r.mapcalc "lsfac0.30m=if(lsfactor.30m==0.000,0,1)"
r.report -n lsfac0.30m unit=p
 [...]
 |0| . . . . . . . . . . . . . . . . . . . . . . . | 20.03|
 |1| . . . . . . . . . . . . . . . . . . . . . . . | 79.97|

g.region rast=lsfactor.15m
r.mapcalc "lsfac0.15m=if(lsfactor.15m==0.000,0,1)"
r.report -n lsfac0.15m unit=p
 [...]
 |0| . . . . . . . . . . . . . . . . . . . . . . . |  3.99|
 |1| . . . . . . . . . . . . . . . . . . . . . . . | 96.01|

g.region rast=lsfactor.10m
r.mapcalc "lsfac0.10m=if(lsfactor.10m==0.000,0,1)"
r.report -n lsfac0.10m unit=p
 [...]
 |0| . . . . . . . . . . . . . . . . . . . . . . . |  2.71|
 |1| . . . . . . . . . . . . . . . . . . . . . . . | 97.29|
```

The new binary layers have 0 in those areas where the *LS* = 0.000 (with floating point accuracy) and 1 everywhere else. You can see that there is a significant difference in the spatial extent of zero areas between the original map (20.03% !) and the re-interpolated map (3.99%). Again, comparison with the percent area with *LS* = 0.000 for `lsfactor.10m` (2.71%) demonstrates that reinterpolation to 15m resolution brings the results much closer to the ones obtained from more accurate 10m resolution DEM. You can also run `d.histogram` for `lsfactor.15m` and `lsfactor.30m` and see how the lower values of erosion are lumped into the zero *LS* factor when the original data were used. This is the impact of insufficient vertical resolution, which causes the topography in low

slope areas being represented by plateaus with zero slope. Using the result directly derived from the original DEM would have significantly underestimated the spatial extent of area with topographic potential for erosion; therefore, we will continue our computation and analysis using the `lsfactor.15m` map layer.

If you display the *LS* values higher than 40.0 using:

```
d.rast lsfactor.15m val=40-400
```

you will see that these high values cover only very small, narrow areas with concentrated flow. In these areas (hollows, upland valleys), values are substantially higher than those typical for the traditional field application of USLE, because they indicate a risk for gully erosion, not modeled by traditional USLE. You can also see that the mountains have the highest topographic risk of erosion due to steep slopes and a high density of concentrated flow areas. However, the mountains have also the largest proportion of forest, so we will now compute the full equation, including the cover factor *C*, to see how much the forest cover compensates for the high topographic risk.

12.1.2 Estimating R, K, and C factors

The *R* factor is not spatially variable in our study area and we can use a constant *R = 65* (you can find the values in any USLE handbook or a related textbook such as Haan et al., 1994). The soil erodibility factor *K* is already included in the Spearfish data set as a raster map `soils.Kfactor`. You can check its values by running `r.report soils.Kfactor` and you will see that the K values are stored as category labels ranging from 0.10 to 0.43 for categories 1 through 8.

The *C* factor is based on land cover. It can be derived from the 100 m resolution vegetation map `vegcover` by editing the category labels. A more detailed *C* factor map can be created from the new 30 m resolution land cover map. It has larger number of classes, so using `r.recode` will be more efficient. The *C*-factor values for different types of land cover can be obtained from literature (Haan et al., 1994).

When using the "category approach" (more typical for GRASS4.*, when only integer values were allowed for raster map layers), copy the map `vegcover` to `cfactor.100` and modify the category labels:

```
g.copy vegcover,cfactor.100m
r.support
 Enter name of raster file for which you will create/modify
 support files
 [...]
 > cfactor.100m
 Edit the header for [cfactor.100m]?  n
```

```
Update the stats (hist.,range) for [cfactor.100m]? (y/n) [n] n
Edit the category file for [cfactor.100m]?  y

The current value for the highest category in [cfactor.100m] is
shown below. If you need to change it, enter another value

Highest category: 6
```

Do not change the number of categories and proceed to the next step using <ESC> <ENTER>. You can now change the TITLE and replace the category labels representing the vegetation by the *C* factor values:

```
TITLE:  Vegetation Cover_____________________________________
CAT   NEW CATEGORY NAME
NUM
 0   no data_____________________________________________
 1   irrigated agriculture_______________________________
 2   rangeland___________________________________________
 3   coniferous forest___________________________________
 4   deciduous forest____________________________________
 5   mixed forest________________________________________
 6   disturbed___________________________________________
```

Retype the TITLE and labels to:

```
TITLE:  C_factor_____________________________________________
CAT   NEW CATEGORY NAME
NUM
 0   0 ______________________________________________________
 1   0.2_____________________________________________________
 2   0.1_____________________________________________________
 3   0.0008__________________________________________________
 4   0.0005__________________________________________________
 5   0.0005__________________________________________________
 6   0.5_____________________________________________________
```

Leave this screen with <ESC><ENTER>, and move on using <ENTER> on all remaining questions. Now you have a *C* factor map with raster cells containing the value of the category number and with each category labeled with *C* factor. The *P* factor is not available, so we can assign it a value of 1.0 and ignore it in our further computations.

Recently, USGS started to provide seamless National Land Cover Dataset on-line, so it was possible to download a much more detailed, 30 m resolution land cover map for Spearfish. We have included it in the Spearfish data set as a raster map layer `landcover.30m`. You can use this map to create a refined *C* factor layer by receding the land cover classes to *C* factor values as follows (run `r.report` to see the land cover associated with each class):

This page intentionally left blank

```
g.region -p rast=landcover.30m
d.erase
d.rast landcover.30m
d.rast.leg -n landcover.30m
r.recode landcover.30m out=cfactor.30m
> 11:23:0.:0.
> 31:32:0.8:0.8
> 41:41:0.0005:0.0005
> 42:42:0.0008:0.0008
> 43:43:0.0005:0.0005
> 51:71:0.001:0.001
> 81:81:0.01:0.01
> 82:83:0.2:0.2
> 85:85:0.001:0.001
> 91:92:0.:0.
> end
```

The individual categories are described in more detail in the metadata file provided with the land cover map; therefore, more accurate values of the *C* factor can be assigned. To get a nice map, we define a special color table:

```
r.colors cfactor.30m col=rules
> 0. blue
> 0.0005 0 180 0
> 0.0008 0 100 0
> 0.001 green
> 0.01 100 255 0
> 0.2 orange
> 0.8 brown
> 1.0 black
> end

d.rast cfactor.30m
```

You can compare the new *C* factor with the one derived from the 100 m resolution `vegcover` data to appreciate the higher quality of the new data.

12.1.3 Computing and analyzing erosion risk

Finally, we can calculate the erosion risk for the given vegetation cover using `r.mapcalc`. In the map layers `soils.Kfactor`, `cfactor.100m`, we use the representation of *K* factor and *C* factor as category labels and we have to put "at" (@) in front of the map name so that `r.mapcalc` uses the category labels instead of category numbers:

```
g.region -p rast=lsfactor.15m
r.mapcalc "erosion.15m=65. * @soils.Kfactor\
                * lsfactor.15m * @cfactor.100m"
```

The resulting map layer represents soil erosion rate at the center of each grid cell in ton/(acre.year). Note that GRASS has automatically resampled the rasters representing the soils and the *C* factor from their original resolutions (20 m and 100 m) to 15 m resolution. Because these two map layers represent discrete homogeneous areas, identified by their integer category numbers (and not continuous fields), such resampling is appropriate.

Similarly as for the *LS* factor, it is useful to define a special color table and categories for the resulting map, while keeping in mind the skewed distribution of erosion rates, with most areas within the low rates range and very few areas in a large interval of high values:

```
r.colors erosion.15m col=rules
> 0 200 255 200
> 1 yellow
> 5 orange
> 10 red
> 20 magenta
> 100 violet
> 1500 black
> end

d.rast erosion.15m
r.support
 [...]
 Edit the category file for [erosion.15m]? (y/n) [n] y
 There are no predefined fp ranges to label
 Enter the number of fp ranges you want to label 0____
```

Now type 7 over 0 followed by <ESC><ENTER>, and you will see the category table with floating point ranges all set to zero. Overwrite the zeros with your selected values and add the category labels:

```
The fp data in map erosion.15m ranges from 0.00 to 699.056291
[erosion.15m] ENTER NEW CATEGORY NAMES FOR THESE CATEGORIES
        65.*@soils.Kfactor*lsfactor.15m*@cfactor.100m________

TITLE: Erosion
FP RANGE            NEW CATEGORY NAME
 0_______-1_______ stable_____________________________________
 1_______-5_______ low________________________________________
 5_______-10______ moderate___________________________________
 10______-20______ high_______________________________________
 20______-50______ severe_____________________________________
 50______-100_____ extreme____________________________________
 100_____-1500____ exceptional________________________________

       Next range number: end__  (of 7)

<ESC><ENTER>
Category file for [erosion.15m] updated
```

You can then run the histogram and report:

```
d.erase
d.histogram -C erosion.15m
r.report -Cn erosion.15m unit=p,k
 |-----------------------------------------------------|
 |                  Category Information   |   %  |  square  |
 |       #|description                     | cover|kilometers|
 |-----------------------------------------------------|
 |     0-1|stable . . . . . .  .| 61.49|161.694225|
 |     1-5|low. . . . . .  .| 15.14| 39.808800|
 |    5-10|moderate . . . . . .  .|  9.34| 24.571575|
 |   10-20|high . . . . . .  .|  7.57| 19.893375|
 |   20-50|severe . . . . . .  .|  5.22| 13.720500|
 |  50-100|extreme. . . . . .  .|  0.87|  2.282400|
 |100-1500|exceptional. . . . . .  .|  0.37|  0.972675|
 |-----------------------------------------------------|
 |TOTAL                                   |100.00|262.943550|
 +-----------------------------------------------------+
```

The report shows that in spite of the very high topographic erosion risk, most of the Spearfish area is quite stable with over 76% area at relatively low erosion risk. However, there is still about 15% area which has high potential for erosion. You can find where those areas are and what their land use is by creating a land cover map for hot spots using `r.mapcalc`:

```
r.mapcalc "erosionrisk.veg=if(erosion15.m>20.,vegcover,null())"
r.colors erosionrisk.veg rast=vegcover
d.rast erosionrisk.veg
d.legend -sm vegcover
```

If erosion is above 20 ton/(acre.year), then the raster value from vegetation map layer will be copied to the new raster, all other cells will be assigned the NULL. The result is a land cover map for areas with high erosion risk and NULLs elsewhere. You can see that the forest dramatically lowers the erosion risk making the mountainous area mostly stable, except for areas with rangeland. You can also compute statistics for your `erosion.15m` map layer:

```
r.univar erosion.15m
 [...]
 Number of cells (excluding NULL cells): 1168638
 Minimum: 0
 Maximum: 699.0562907236
 Range: 699.056
 Arithmetic mean: 5.02223
 Variance: 211.117
 Standard deviation: 14.5299
 Variation coefficient: 289.311 %
```

While the range of values is extremely high, the mean erosion for this area is relatively low, confirming that the exceptionally high erosion risk applies to only a very small number of grid cells (mostly disturbed areas on steep slopes over 25°). Note that your numbers may be slightly different because of small differences in DEM due to its interpolation from random sites and potential future updates to the Spearfish data set.

To compute a more detailed and up-to-date erosion risk map, you can change your region to the raster map `lsfactor.10m` and re-run the computation using the *LS* factor and *C* factor based on the new data.

This example demonstrated an application of GRASS for regional scale natural resource management task by assessing erosion risk and identifying critical areas where land use change may be desirable.

12.2. GIS MODELING FOR LAND MANAGEMENT (⇑)

In this section you will be a GIS expert for a team preparing sustainable land management plan for an experimental farm in Wake county, North Carolina. You will be asked to provide comprehensive geospatial support for your colleagues who will be making decisions and creating plans for this area.

Our study location faces development pressures from the neighboring metropolitan area. At the same time there is a strong support of the current community for preserving open spaces and the rural character of the area. Water from the farm goes to a water reservoir which serves as a backup for drinking water for the neighboring city, making water pollution prevention a key land management issue.

To fulfill this task you propose to do the following:

- Build a GIS database for the study area:
  - search the Internet, contact local agencies to find what data are available;
  - decide which coordinate system will be used for the core LOCATION used for modeling and analysis;
  - set up the necessary LOCATION(s), import and project data to common coordinate system;
  - evaluate the quality of data, identify missing information and needs for additional mapping and/or monitoring together with other members of the team;
- Analyze the data and derive additional map layers needed for the project:
  - create raster DEM from contours, add buildings for 3D view;

- derive slope, aspect, and curvatures describing the terrain geometry;
- delineate watersheds and flow related parameters;
- create high resolution land cover map;

- Analyze the current land use, identify potential problems and solutions:
  - identify areas unsuitable for crop production;
  - find suitable locations for selected crops;
  - evaluate erosion and sedimentation risks;
  - assess need for conservation measures;
  - evaluate the impact of potential development;
- Communicate the results: create maps, views and put data on-line.

While we cannot describe all these tasks in detail, we will show how to do some of them and you can try to find out how to do the ones that sound interesting by yourself.

12.2.1 Building the GIS database

Building the database is usually the most time consuming and sometimes frustrating part of GIS work, no matter what kind of software you use. However, once the data are gathered and nicely integrated in a consistent manner, the work with geospatial data may become quite exciting and rewarding. The task of searching and acquiring the georeferenced data is becoming easier due to the rapid development of WebGIS technology. For example, several data sets needed for this project are available from the Wake county GIS site[3]. Our study area is covered by tiles 0791 and 0792 that can be accessed directly from the related Map Section.[4] We have downloaded the data from the Wake county GIS and other sources, and prepared a ready to use database available from the GRASS Tutorials Web site[5] as `wake_spft_new.tar.gz`. After downloading and installing it (in a similar way as we did for Spearfish) we can start to explore the data that are available.

First, we need to check the projection, units, and region that have been used for the database:

```
g.projinfo
 PROJ_INFO file:
 name:     State Plane
 datum:    nad83
[...]
 proj:     lcc
 a:        0.63782064e+07
 es:       0.6768657997291094e-02
 x_0:      0.6096012192024384e+06
 y_0:      0
```

```
 lon_0:    79dw
 lat_0:    33d45'n
 lat_1:    36d10'n
 lat_2:    34d20'n
 ----------------------------------------------------------
 PROJ_UNITS file:
 unit:     foot
 units:    feet
 meters:   0.3048000000

g.region -p
 projection: 99 (State Plane)
 zone:       0
 datum:      nad83
 ellipsoid:  a=6378206.4 es=0.006768657997291094
 north:      728002
 south:      719500
 west:       2089600
 east:       2100604
 nsres:      6
 ewres:      6
 rows:       1417
 cols:       1834
```

Many counties in US still use feet for their GIS, so to minimize the need for data projection, the database was set up in the State Plane NAD83 coordinate system with feet as units. This will allow us to discuss some issues related to the use of units different from meters in GRASS. Note that in GRASS 5.3, information about the horizontal datum is included in the `PROJ_INFO` file.

For our project, we will need the base map data such as elevation, surface water (streams and lakes), roads, railroads, and land use/land cover. We may also need thematic map layers important for land management, such as soils or climate data. By running the `g.list` command we can find out which data layers are available and then evaluate whether they are suitable for our project. We will now look at some of the given data in more detail.

Elevation. First, we explore the possibility of using the elevation data `elev.90ft` from the USGS National Elevation Data set (NED) which merges the best elevation data available into a seamless, approximately 90 ft (30 m) resolution raster format. The data were downloaded from the USGS NED web site in geographic coordinates and projected into the State Plane, feet coordinate system. To evaluate the data quality, we compute and display the slope and aspect and then view the data in 3D using `nviz`:

```
g.region rast=elev.90ft -p
r.slope.aspect elev.90ft slo=slope.90ft asp=aspect.90ft \
               zfac=0.3048
```

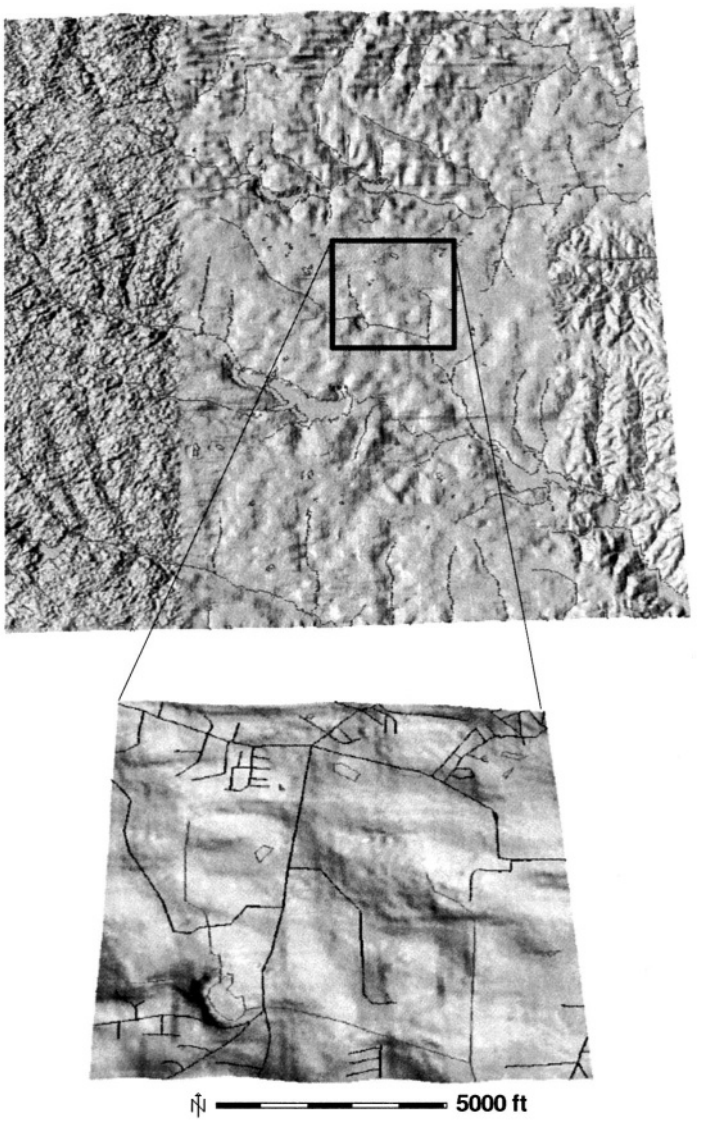


Figure 12.2. Sample from the USGS seamless National Elevation Data set (elevation map with vector roads overlayed) showing insufficient resolution in our study area and overall variable quality, revealed when viewing in `nviz`

```
d.rast aspect.90ft
d.rast slope.90ft
nviz elev.90ft
```

When computing slope in a LOCATION with units set to feet, it is important to use `zfac=0.3048` to obtain the correct values of slope. GRASS internally transforms the distance to meters and it is the user's responsibility to transform the elevation values to meters, too; otherwise, the slopes will be severely exaggerated. The slope and aspect maps, as well as Figure 12.2 created with `nviz`, show that in our study area, the NED elevation model has low resolution, and within the region it has uneven quality. While this elevation data would be appropriate for a regional scale study, they are not suitable for local land management.

Wake county provides topographic data in the form of 2 ft contours as vector data in ESRI SHAPE and E00 format upon request. We have acquired the data and included them in the provided data set. To view the contours run:

```
g.region vect=lw_contour -p
d.vect lw_contour col=brown
```

The contours provide a detailed description of topography; however, for most of our applications, we will need elevation data as a raster DEM. We will create it using spatial interpolation in the next Section 12.2.2.

Recently, LIDAR-based elevation data became available for this area in the form of bare ground point data, 20 ft grid and 50 ft grid with stream enforcement. The data can be downloaded from the North Carolina Flood Mapping Program[6].

Roads, buildings, streams, soils. Additional vector data are line features representing roads, streams and lakes and building footprints. We will display them using `d.vect`:

```
d.vect lw_roads col=black
d.vect lw_hydro col=blue
d.vect lw_buildings col=grey
```

The soil data are available as a polygon map layer. After running `v. support`, we transform them to a soil raster map at 6 ft resolution:

```
d.erase
d.vect lw_soil
v.support -r lw_soil
g.region res=6 vect=lw_soil -p
v.to.rast lw_soil out=lw_soil.6 row=2000
d.rast lw_soil.6
```

We get a nice map; however, if we run `r. report lw_soil. 6` we can see that the map includes only polygon numbers. To create raster maps with properties that we will need for modeling and analysis, we have to make a new copy

of this map for each attribute and then copy its vector category file to the new raster category file (see Section 4.2.1 and Section 5.1 for more details about categories). You can check which attributes were included with the soil data layer by listing the files in the vector categories subdirectory `dig_cats` in the related MAPSET directory:

```
ls

 lw_soil.awch       lw_soil.laydepl   lw_soil.phh
 lw_soil.awcl       lw_soil.layernum  lw_soil.phl
 lw_soil.bdh        lw_soil.musym     lw_soil.poly#
 lw_soil.bdl        lw_soil.omh       lw_soil.surftex
 lw_soil.compname   lw_soil.oml       lw_soil.texture
 lw_soil.kfact      lw_soil.permh     lw_soil.wtdeph
 lw_soil.laydeph    lw_soil.perml     lw_soil.wtdepl
```

The attributes are described in the NRCS documentation.[7] If you plan to do substantial work with multiattribute data, it is worth considering the link with databases such as PostgreSQL or another DBMS linked through the ODBC interface. This functionality has been substantially improved in GRASS 5.7, the development version. Here, however, we will work only with GRASS 5.3 tools.

To create a raster map layer with the names of soils using the category file `lw_soil.musym`, we run (from within the appropriate MAPSET directory):

```
g.copy rast=lw_soil.6,lw_soil.musym
cp ./dig_cats/lw_soil.musym ./cats
d.rast lw_soil.musym
r.report lw_soil.musym unit=a,p
```

A raster map layer for the *K* factor can be generated from `lw_soil.kfact` file in a similar way:

```
g.copy rast=lw_soil.6,lw_soil.kfact
cp ./dig_cats/lw_soil.kfact ./cats
d.rast lw_soil.kfact
r.report lw_soil.kfact unit=a,p
```

If we need additional raster maps of soil properties, we just create a copy of the raster `lw_soil.6` with the appropriate name and copy the relevant category file to the raster `cats` directory as shown above.

Land cover. Land cover data are usually derived from satellite or aerial imagery. For our area, 50 ft (16 m) resolution Land use/Land cover (LULC) data and 90 ft (30 m) resolution National Land Cover data are available. Both are derived from satellite imagery for regional scale studies. They do not have the

level of detail needed for local planning, for example, the individual fields or service roads are not represented.

To get more detailed information about the land cover, we downloaded and imported the 1 ft resolution Infrared Digital Orthophoto Quarter Quadrangles (IR-DOQQ, from the Wake county GIS, see URL in Endnotes). The imported images can be used to digitize the individual fields, roads, ponds and other features and to create a high resolution land use map layer. We have imported such a land use map as polygon vector data; you can display it and transform it to a raster as follows (remember that you need to be in an appropriate MAPSET directory to copy the category files):

```
d.vect lw_landcov
v.to.rast lw_landcov out=lw_landcov
d.rast lw_landcov
g.copy rast=lw_landcov,lw_landcov.type_id
cp ./dig_cats/lw_landcov.type_id ./cats/lw_landcov.type_id
d.rast lw_landcov.type_id
r.mapcalc "lw_landcov.type=int(@lw_landcov.type_id)"
d.rast lw_landcov.type
```

We have created two raster maps for land cover. The values stored in `lw_landcov.type_id` are unique category numbers for each field, with category labels representing a number associated with a particular land use. This map can be used for creating new management scenarios by assigning different types of crops or land use to the individual fields. Because many fields have the same land use type, we have created a simplified raster map `lw_landcov.type` using `r.mapcalc`. In this map, the values represent the land use type; we assign them description as a category label using `r.support`. You can check the land use categories and their area as follows:

```
r.report lw_landcov.type unit=a,p
|-----------------------------------------------------|
|         Category Information          |          |  %  |
| #|description                         |     acres| cover|
|-----------------------------------------------------|
| 1|water. . . . . .  |  0.213218|  0.09|
| 2|forest . . . . . | 16.333529|  6.89|
| 3|building . . . . . |  2.715643|  1.15|
| 4|bare soil. . . . . |  0.782628|  0.33|
| 5|asphalt road . . . . . |  2.603249|  1.10|
| 6|gravel road. . . . . |  3.309019|  1.40|
| 7|grapes . . . . . |  1.065266|  0.45|
| 8|agriculture. . . . . | 54.097169| 22.83|
| 9|wetland. . . . . |  0.842130|  0.36|
|10|dirt road. . . . . |  0.354538|  0.15|
|11|grass. . . . . | 56.086382| 23.67|
| *|no data. . . . . | 98.550746| 41.59|
|-----------------------------------------------------|
|TOTAL                                  |236.953517|100.00|
```

12.2.2 Deriving new map layers

The basic data sets provide foundation for the study of landscape, its properties, and spatial relations between its features. To analyze the landscape and its processes, we need to derive additional map layers which provide information about the landscape properties in the form needed for models and decision making.

Computing the DEM and basic topographic parameters. While the contour data are a standard for representation of topography on paper maps, spatial analysis and modeling can be done more efficiently with a raster DEM. To compute the raster DEM we can transform the points defining the contour lines to sites and use `s.surf.rst`. It allows us to simultaneously compute the topographic parameters slope, aspect and curvatures that we will need later both for planning and modeling. First we set the region to our study area, defined by the soil map layer, and the resolution to 6 ft, which is enough to capture the roads and dams. Then we can try to run `s.surf.rst` with the default parameters and include the profile curvature in the output. As we have mentioned in the Section 6.4.3, interpolation from contours often leads to a surface with waves along contours. This can result, for example, in a false pattern of erosion and deposition (see Mitas and Mitasova, 1999) and it is useful to check this by displaying profile curvature along with contours as shown in the Figure 12.3.

```
g.region res=6 vect=lw_soil -p
v.to.sites -ad lw_contour out=lw_contour
s.info lw_contour
s.surf.rst lw_contour elev=elev.6 pcu=pcurv.6 &
d.rast pcurv.6
d.vect lw_contour
```

As you can see, for this data set, the tension is too high and the curvatures follow the pattern of contours (Figure 12.3). In some locations, we can even see the triangles which were used to derive the contours. The geometry of the resulting surface thus reflects the geometry implied by the distribution of data points rather than real terrain. To minimize this bias we can lower the tension, reduce the density of points, and introduce slightly higher smoothing:

```
s.surf.rst lw_contour elev=el.6t10 slope=sl.6t10 asp=as.6t10 \
               pcu=pc.6t10 tcu=tc.6t10 dmin=9 ten=10 smo=0.8 &
d.rast pc.6t10
d.vect lw_contour
nviz el.6t10
```

We will lose some detail, but we will also reduce the artificial patterns, as you can see by overlaying the profile curvature with contours. We can explore the results by displaying the DEM in `nviz` and draping the computed topographic parameters and contours over it.

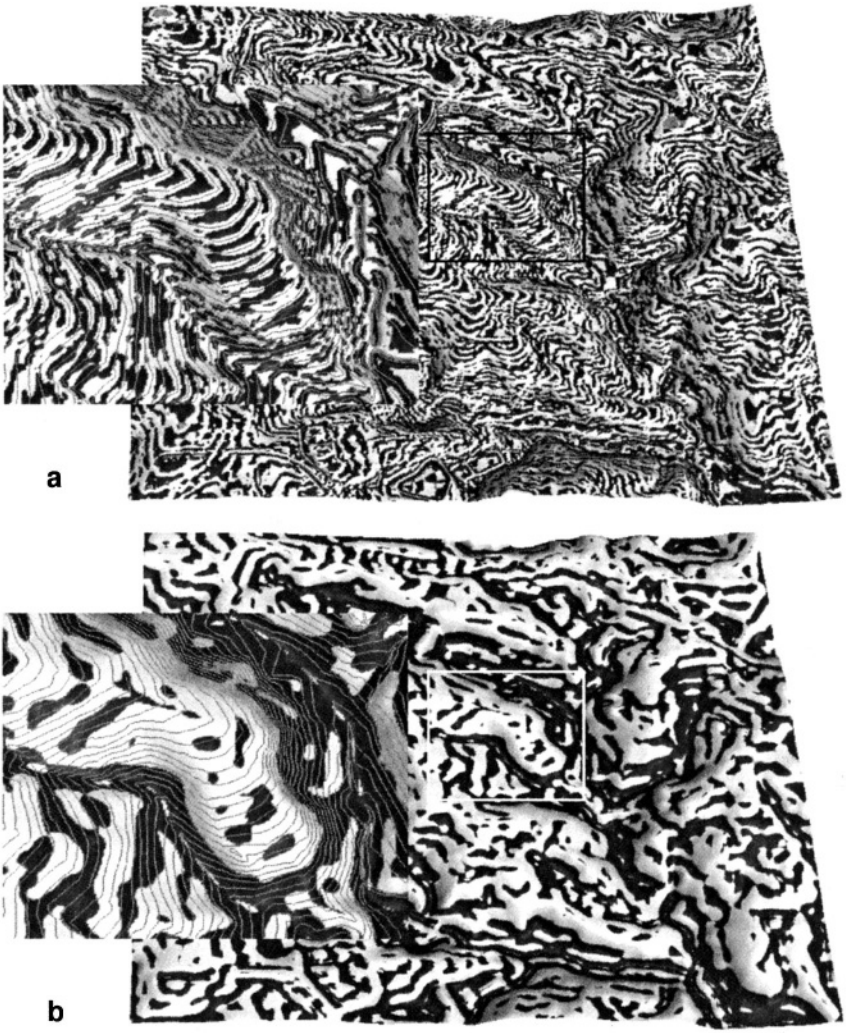


Figure 12.3. Interpolating DEM from contours: profile curvature displayed with input contours a) tension is too high and the pattern of curvature follows contours, b) with lower tension the pattern disappears

For some applications it is useful to create a DEM with buildings. We can compute it using the building footprints which are available as vector data. First, we need to transform them to raster and then we add their average height, in our case 20 ft, to the DEM:

```
v.to.rast lw_buildings out=buildings
d.rast el.6t10
d.rast -o buildings
r.null buildings null=0
r.mapcalc "elevbldg.6=if(buildings > 0, el.6t10+20, el.6t10)"
r.mapcalc "elevbldgco.6=if(buildings > 0, -1, el.6t10)"
r.colors elevbldgco.6 rast=el.6t10
nviz elevbldg.6 col=elevbldgco.6 vector=lw_roads,lw_hydro
```

Note that we had to convert NULLs in the `buildings` map to zeroes first because NULLs override any other value when used in `r.mapcalc` expressions (see Section 5.2). We have also created an additional raster to be used for color and displayed it together with roads and streams in 3D using `nviz` (Figure 12.4). The surface has the typical elevation color scheme with buildings displayed in white with grey shades. You can adjust the z-exaggeration and light to enhance the 3D perception and save your view as a PPM, TIFF or RGB image (see Chapter 8).

Topographic analysis. Topography has a profound influence on fluxes in landscapes and is often a major factor in decisions related to land use. Therefore map layers representing watershed boundaries and topographic parameters (see Section 5.4) are used as the basis for selection of land management units,

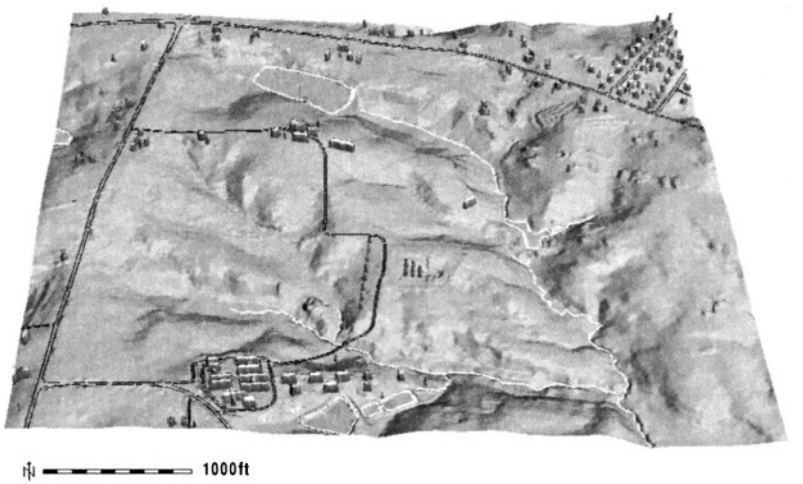


Figure 12.4. High resolution DEM interpolated from 2 ft contours with buildings viewed in `nviz`. Our practice area is in the west section

placement of monitoring equipment, evaluation of suitability of the current land use for existing topographic conditions and a number of other issues.

Local (point) topographic parameters. A number of parameters describing the geometry of the terrain surface are needed as inputs for land use suitability analysis and modeling of various processes. These parameters represent measures of change in elevation (gradient) and measures of rate of this change (curvatures) and are usually expressed as the following parameters (see also Section 5.4.5 and the exact mathematical definitions and equations in the Appendix B.3):

- slope (steepest slope angle, magnitude of gradient);
- aspect (slope orientation, direction of gradient, steepest slope direction, direction of flow);
- profile curvature (curvature in the direction of the steepest slope – perpendicular to contour lines, convex curvature leads to accelerated and concave to decelerated flow);
- tangential curvature (curvature in the direction of contour tangent – perpendicular to the steepest slope direction, convex curvature leads to dispersal and concave to convergent flow).

It is possible to define a number of additional parameters (see, for example the module `r.param.scale`), however, for most applications only slope and aspect is needed.

We have already computed these parameters simultaneously with interpolation using `v.surf.rst`, and we have used the profile curvature to look for artificial patterns in the DEM (Figure 12.3). We have also used `r.slope.aspect` to derive slope and aspect from the USGS NED data to evaluate potential problems in this DEM (see the relevant paragraph above).

Because the landscape processes have multiscale character and different processes are dominant at different scales, it is sometimes useful to extract topographic features and patterns at different levels of detail. In `s.surf.rst` or `v.surf.rst`, this can be achieved by changing tension and smoothing (lower tension and higher smoothing produces smoother topography with curvatures representing main features, higher tension captures more detail and smaller features, see Section 7.3 as well as the Figure 12.3). We can also use `r.param.scale`, which has additional options to extract the main topographic features, peaks, ridges, passes, channels, pits and planes and compute a number of additional parameters.

Learn more about topographic analysis in the book by Wilson and Gallant, 2000, and publications by Wood, 1996, Mitasova and Hofierka, 1993, and Moore and Burch, 1986.

Watersheds and flow parameters. Watersheds (also called basins or catchments) are defined as the areas draining into a single point. They are a suitable choice as landscape units for land use and natural resource management. Output of such units, in terms of water discharge, sediment, and pollutant loads can be relatively easily monitored, providing a quantified measure of impact of land management actions.

Because each point of landscape has a watershed associated with it (it can be anything from the point itself for a peak to hundreds of square miles for outlets of big rivers), we need to select a suitable size for our watershed management units. For our study area, a 50000 grid cells watershed size can be used as a threshold parameter for `r.watershed`. We also compute a flow accumulation map and a drainage map, as they may be needed as inputs for models:

```
r.watershed elev=el.6t10 accum=accum.6 basin=basin.6 \
            threshold=50000 drainage=drain.6 stream=stream.6
d.rast basin.6
r.report basin.6 unit=p,a
d.what.rast
 [...]
 2095334.40430738(E) 722934.27614887(N)
 basin.6 in lakewheeler_hm  (6)
```

The report shows that our basins have areas from 15 to 153 acres. We have used `d.what.rast` to find the number of the watershed that will be the focus of our more detailed study (it has category number 6) and according to our report table it has 62.81 acres. We can create a vector representation of watershed boundaries (Figure 12.5) using `r.poly` so that we can overlay them over other data such as imagery, or transfer them to GPS to guide mapping activities:

```
r.poly basin.6 out=basin.6
d.vect basin.6 col=red
v.support -r basin.6
d.vect.labels basin.6 col=black
```

To view the accumulation map, you need to create a non-linear color table for `accum.6` as explained in the previous Section 12.1, or display only the cells with high values, for example:

```
d.rast accum.6 cat=10000-400000
```

Remember that `r.watershed` uses the D8 algorithm, so it is not suitable for hillslope flow pattern (Figure 12.5 a, b). It "flows" through all depressions (unless they are explicitly given as input), so the stream network will be fully connected and drain into the outlet. We can get the coordinates for the watershed outlets using `d.what.rast`:

```
d.rast basin.6
d.vect lw_hydro
d.what.rast
 [...]
 2096054.56(E) 721547.96(N)
 basin.6 in lakewheeler_hm  (6)

 2097854.97(E) 721043.84(N)
 basin.6 in lakewheeler_hm  (4)
 [...]
```

and then input the outlet coordinates into GPS for evaluation of the suitability of these locations for monitoring directly in the field. It is highly probable that the exact points derived from the watershed map won't be ideal for monitoring and new locations may be selected. It is then necessary to derive the modified watershed boundaries for the exact locations of the monitoring stations using `r.water.outlet`. In our study area, we are installing two monitoring stations: one at the point A (Figure 12.5) near the road above a constructed wetland (2095700.00, 722600.00) and one at the point B which is the outlet for our entire study area. The first station is far from the outlet of the watershed number 6; therefore, we need to delineate a new one. Because the `stream.6` map layer does not show a stream in our subarea (the threshold for streams is higher, as we can see by displaying `lw_hydro` vector map over `stream.6`), we need to use the accumulation map to compare the field monitoring location with the concentrated flow defining the outlet. We find that it is shifted by a few meters and there are only few cells "upflow" from this point. Also at 6 ft resolution, the flow from `r.watershed` is split into many parallel lines (Figure 12.5 a), therefore we need to decrease resolution to get a single, clearly defined outlet. We can then find new coordinates for outlet using `d.what.rast` and define a new watershed. The entire procedure is as follows:

```
g.region res=20 -p
r.watershed elev=el.6t10 accum=accum.20 basin=basin.20 \
            threshold=2000 drainage=drain.20 stream=stream.20
r.colors accum.20 rast=accum.6
d.what.rast
 [...]
 2095648.79956928(E) 722620.03377117(N)
 [...]
r.water.outlet drainage=drain.20 basin=basin.A20 \
                        east=2095648 north=722620
d.rast basin.A20
g.region res=6
r.poly -l basin.A20 out=basin.A20
d.vect basin.A20 col=black
```

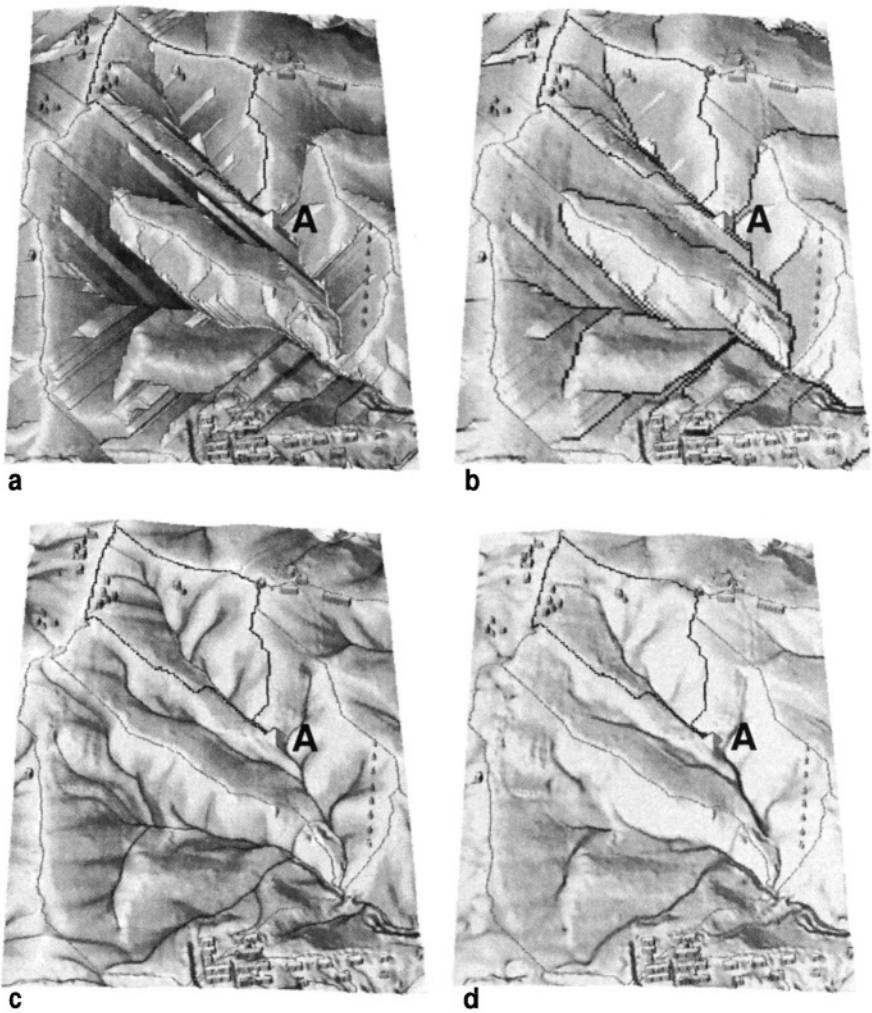


Figure 12.5. Flow accumulation maps based on a) D8 method (`r.watershed`) 6 ft resolution, b) D8 method (`r.watershed`), 20 ft resolution, c) vector-grid (D-infinite, `r.flow`) 6 ft resolution, d) multiple directions flow (`r.topidx`). Watershed boundaries as derived from `r.watershed` and `r.water.outlet` for the monitoring station A are shown as vector lines draped over the DEM

The flow accumulation map generated at 20 ft resolution has better defined concentrated flow areas, making the delineation of the watershed above the monitoring station A feasible (Figure 12.5 b). Note that for some hydrologic and pollutant transport models, we could further divide the study area into

smaller sub-watersheds and half-watersheds using the `r.watershed` option `half.basin` for this purpose.

To derive a spatial better description of overland flow, we can use `r.flow`. The module uses the vector-grid (D-infinite) algorithm; therefore, it can better handle flow routing at high resolution. First, we set the region to our 6 ft resolution DEM `el.6t10`; then we use `r.flow` to compute raster maps of flow accumulation and flow path length as well as flow lines vector map:

```
g.region rast=el.6t10
r.flow el.6t10 flout=flowline.6 lgout=flowlg.6 dsout=flowacc.6
d.rast flowacc.6
```

When comparing the flow accumulation `flowacc.6` with `accum.6` derived from `r.watershed`, we can quickly see the more realistic and more detailed description of flow in our new result (Figure 12.5 c). The flow path length map `flowlg.6` provides information about the length of the flowline drawn from each cell. It can be used to compute the longest flow path for a given watershed, a parameter needed in hydrologic models for estimation of time to steady state (how long it will take for all water in the watershed to reach the outlet). For our study subarea, we can estimate this distance by displaying the `flowlg.6` map overlayed with the vector `flowline.6` map and using `d.what.rast` to find the flowline length values at the monitoring station and at the farthest point of the watershed and then subtract those two values:

```
d.rast flowlg.6
d.vect flowline.6 col=black
d.what.rast
  [...]
  2095676.48172339(E) 722592.19867314(N)
  flowlg.6 in lakewheeler_hm, actual  (1197.761963)

  2094644.24811718(E) 724218.56702265(N)
  flowlg.6 in lakewheeler_hm, actual  (3213.695557)
```

After subtraction, the value is 2016 ft (about 600 m). When using the output from `r.flow`, it is important to know that the module stops flow routing when slope is 0.0, so it is suitable for estimation of flow on hillslopes, smaller watersheds, or DEMs without pits or flat areas. To see where the flow stopped, it is useful to display the flow accumulation as a surface using `nviz flowacc.6`. For comparison, we can also extract the flow accumulation map from the saturated flow index generated by the module `r.topidx`, which uses the multiple directions flow algorithm capable of representing dispersal flow (Figure 12.5 d).

12.2.3 Land use analysis, problems and solutions

We will use our baseline data and derived parameters to answer some questions that are relevant to land use planning. Because there is a large number of possibilities for combining the data to get answers to specific problems, we show only few examples.

Locate areas unsuitable for crop production. With input from our agriculture engineering colleagues we define the areas unsuitable for selected crops using the following conditions:

- slope greater than 7°;
- distance to streams and ponds less than 60 ft, to prevent the pollution of water with nutrients and chemicals applied to the fields;
- distance to roads 15 ft to provide space for road maintenance and prevent pollution of crops from traffic.

After transforming the vector representation of roads and streams to raster we use a combination of `r.buffer` and `r.mapcalc` to identify the areas that fulfill the above conditions:

```
v.to.rast lw_hydro out=lw_hydro
r.buffer lw_hydro out=lw_hydro.buff60 dist=60 units=feet
r.null lw_hydro.buff60 null=0
v.to.rast lw_roads out=lw_roads
r.buffer lw_roads out=lw_roads.buff15 dist=15 units=feet
r.null lw_roads.buff15 null=0
```

Note that the result of `r.buffer` is a category map with categories 1 (original data) and 2 (buffers), while the distance values are stored as category labels. We use `r.null` to change unclassed NULL values to zeros, because NULL would override everything else in our map algebra operations:

```
r.mapcalc "noagrlanduse=if(sl.6t10 >= 7 || lw_hydro.buff60 \
                  || lw_roads.buff15, lw_landcov.type, 0)"
r.colors noagrlanduse rast=lw_landcov.type
d.rast noagrlanduse
r.report noagrlanduse unit=p,a
```

The resulting map layer shows the land use which is in the areas that are not suitable for agriculture, that is on steeper slopes or too close to roads and water. We can see that most of these areas are already forested or covered by grass, except for small patches along the roads and on steeper slope. The report provides us with information about the number of acres in different land use classes that are in these locations. For example, we can find that we have only

about 3 acres of agricultural land in unsuitable locations, if the above criteria are used.

You can further explore the effects of various stream buffer widths by using multiple distances for the buffer map layer:

```
r.buffer lw_hydro out=lw_hydro.buff dist=30,60,120,180 uni=feet
```

resulting in a raster map with 5 categories (one for original data and 4 for distances). Use `r.mapcalc` to find out how much of the current agricultural land we may lose if we increase the size of buffers. We can also explore the impact of various definitions of stream. For example, the map layer `lw_hydro` includes only the permanent streams large enough to be included on a map. It does not capture intermittent streams and concentrated flow areas. To include them, we can create an alternative map layer representing all areas with concentrated flow using the flow accumulation map from `r.flow`, repeat the analysis, and compare the results.

Locations suitable for a specific crop. To find the best locations for a specific type of plant within the areas designated as agricultural fields, we need to consider additional conditions, such as soil properties, soil moisture, and solar radiation. While there are complex plant growth models available as external software tools, a quick assessment can be performed using the GRASS modules. For example, the properties of a location suitable for a given crop can be defined as follows:

- land suitable for agriculture (as derived above), within the existing agricultural fields;
- location receives total of at least 2100 kWh/m^2 of global radiation during the growing season;
- location has suitable soil properties, such as pH, and soil texture.

We can review various soil properties by inspecting the related soil attributes stored in the files in the vector categories subdirectory (see the previous Section 12.2.1), and see that many properties, such as pH and texture are spatially homogeneous in our area and their distribution will not affect the selection of a particular location. Therefore, we will focus our analysis on spatial distribution of insolation and the conditions defined as suitable for agriculture in the previous paragraph.

To find the spatial distribution of potentially available global solar radiation we can compute all of its three components: direct, diffuse, and reflected using the module `r.sun`. We will use the latitude 35.5° N found by `g.region -l` and the elevation, slope, and aspect maps computed in the Section 12.2.1 as

inputs (see Section 5.4.5, the manual page for `r.sun` and Appendix B.4 for more details on this module). If the growing season for a specific crop is between March 15 and October 15, we can use a shell script (see Chapter 11) to compute a sum of daily global radiation for this period of the year:

```
#!/bin/sh
echo "Enter elevation map:"
read elev
g.copy rast=$elev,SOLelev
echo "Enter slope map:"
read sl
g.copy rast=$sl,SOLslope
echo "Enter aspect map:"
read asp
g.copy rast=$asp,SOLaspect
echo "Enter Latitude of given region:"
read lat

i=75
lastday=288

#generate an empty map for global radiation:
r.mapcalc "global.rad=0"

while [ $i -le $lastday ]
do
 # generate map names convenient for xganim and r.out.mpeg:
 DAY=`echo $i | awk '{printf "%03i", $1}'`

 echo "Computing radiation for day $DAY. . ."
 r.sun elevin=SOLelev aspin=SOLaspect slopein=SOLslope\
       lat="$lat" day="$i"\
       beam_rad=b_rad.$DAY diff_rad=d_rad.$DAY\
                           refl_rad=r_rad.$DAY

 #add to (cell-wise) global energy:
 r.mapcalc "global.rad=global.rad + b_rad.$DAY +\
                       d_rad.$DAY + r_rad.$DAY"
 r.timestamp b_rad.$DAY date="$i days"
 r.colors b_rad.$DAY col=gyr
 r.timestamp d_rad.$DAY date="$i days"
 r.colors d_rad.$DAY col=gyr
 r.timestamp r_rad.$DAY date="$i days"
 r.colors r_rad.$DAY col=gyr

 i=`expr $i + 1`
done
```

```
#cleanup:
g.remove rast=SOLelev,SOLaspect,SOLslope
echo "Finished."
```

The module outputs all three components of global radiation separately, so we need to add them to get the global radiation. We then use map algebra to find the areas within the agricultural fields where this value exceeds our given threshold, while excluding the areas which were identified as unsuitable for growing crops in the previous paragraph:

```
r.mapcalc "plant=if(global.rad/1000.>2100 && noagri==null \
                                    && lw_landcov.type==8,1,0)
d.rast plant
r.report plant
```

We can use `r. report` to find how large is the area that is suitable for our plant. The resulting time series of radiation maps (e.g. `b_rad.075` through `b_rad.288`) can be animated in `xganim` to get a better insight into evolution of solar radiation pattern:

```
xganim  view1="b_rad.*"
```

The module also allows us to display several maps simultaneously as time series:

```
xganim view1="b_rad.*" view2="d_rad.*" view3="r_rad.*"
```

To animate the result in 3D as a color draped over the DEM along with the vector map layer representing the land use polygons, you can use `nviz` scripting capabilities with the file sequencing tool (see Section 8.2.3 and `nviz` tutorial for more details).

Planning conservation measures using estimate of sediment flow. The flow accumulation map `flowacc.6` indicates a significant potential for negative impact of concentrated flow, which can cause formation of gullies and transport of sediment and pollutants through protective stream buffers. One of the common practices to mitigate the possible impact of concentrated flow are grassed waterways.

We can use a simple combination of topographic analysis and map algebra to identify the areas which may need protection by grassed waterways. This type of management practice requires installation of a grassed path in those areas which have concentrated flow and which are without protective vegetation during certain time of year (for example agricultural fields). First, we find the high risk areas by computing a topographic index for sediment flow, which combines the upslope area (flow accumulation multiplied by resolution) with slope (we have computed both layers in the Section 12.2.2):

```
r.mapcalc "sedflow.6=flowacc.6*6*sin(sl.6t10)"
r.colors sedflow.6 col=rules
 fp: Data range is 0.00 to 26663.21484375
> 0 200 255 200
> 10 yellow
> 50 orange
> 100 red
> 500 magenta
> 30000 violet
> end

d.rast sedflow.6
```

We have assigned a special color table to the new map highlighting the high sediment flow areas. We now extract those areas where the value of the sediment flow index is greater than 50 and display the result over the land use map:

```
r.mapcalc "gullies.6=if(sedflow.6 > 50, sedflow.6)"
r.colors gullies.6 rast=sedflow.6
r.null gullies.6 setnull=0
d.rast lw_landcov.type
d.rast -o gullies.6
```

We can see that we have a high risk area in almost every field. To create a map of proposed grassways we use map algebra to extract the potential gullies in the areas with agricultural land, grapes, and bare soil (categories 8, 4, 7):

```
r.mapcalc "grassway.6=if((lw_landcov.type == 8 || \
                          lw_landcov.type == 4 || \
                          lw_landcov.type == 7) && \
                          gullies.6, 1)"
r.null grassway.6 setnull=0

r.colors grassway.6 col=rules
> 1 green
> end

d.rast lw_landcov.type
d.rast -o grassway.6
r.report grassway.6 unit=p,a
 |-----------------------------------------------------|
 |  Category Information                 |   %   |     |
 |#|description                          | cover |  acres|
 |-----------------------------------------------------|
 |1| . . . . . . . . . . . . . . . . . . |  0.19 |  1.162484|
 |*|no data. . . . . . . . . . . . . . . | 99.81 |596.161891|
 |-----------------------------------------------------|
```

The displayed map shows that almost each field needs two or more grassways, (Figure 12.6). And the report provides us with an estimate of over 1 acre for grassed area within the fields.

Net erosion and deposition modeling using flow divergence. Our previous examples involving erosion processes (both for Spearfish and for grassways) focused on soil detachment. However, the detached soil can be deposited relatively close to the source without causing pollution problems in the streams. Simulating net erosion and deposition under spatially variable topographic and land cover conditions is a complex task and requires external simulation tools. However, a simplified estimate of erosion and deposition pattern can be obtained relatively easily using the concept of flow divergence.

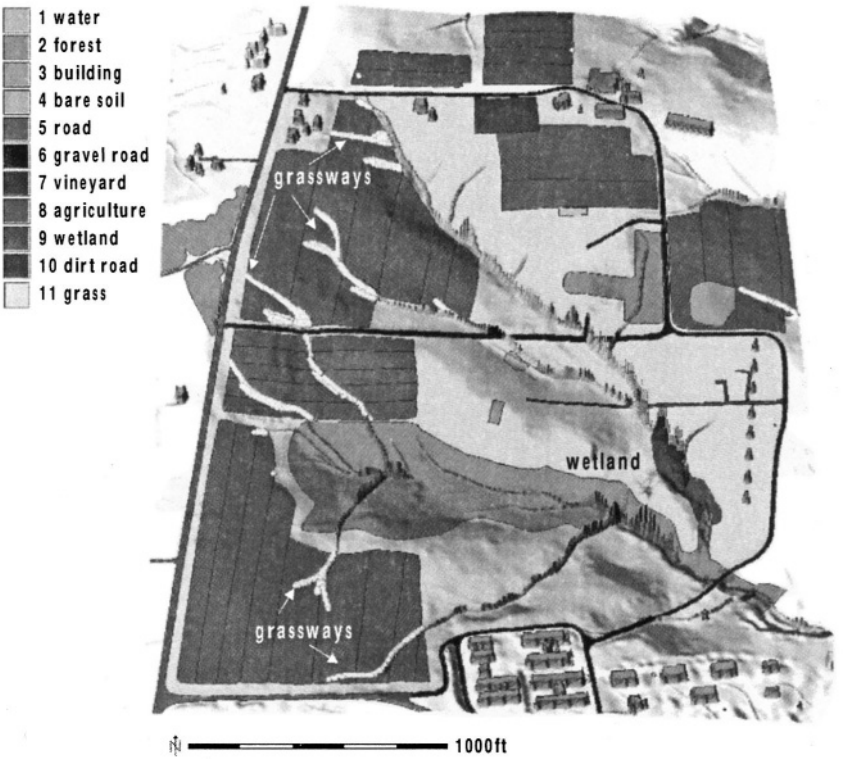


Figure 12.6. Proposed grassways viewed as dense sites draped over DEM with the current land cover displayed as a color map. The DEM has buildings and sediment flow potential added to elevation to enhance the understanding of relation between flow and land use in the watershed

The *Unit Stream Power Based Erosion/Deposition model* (*USPED,* Mitasova et al., 2001) estimates a simplified case of erosion/deposition using the idea originally proposed by Moore and Burch, 1986. It combines the *RUSLE* parameters and upslope contributing area per unit width A to estimate the sediment flow T:

$$T \approx RKCPA^m(sin\beta)^n, \tag{12.3}$$

The net erosion/deposition D is then computed as a divergence of sediment flow:

$$D = \nabla \cdot (T\mathbf{s_0}) = \frac{d(T\cos\alpha)}{dx} + \frac{d(T\sin\alpha)}{dy}, \tag{12.4}$$

where α in degrees is the aspect of the terrain surface (direction of flow). The exponents m,n control the relative influence of water and slope terms and reflect the impact of different types of flow. The typical range of values is $m = 1.0 - 1.6$, $n = 1.0 - 1.3$, with the higher values reflecting the pattern for prevailing rill erosion with more turbulent flow when erosion sharply increases with the amount of water. Lower exponent values close to $m = n = 1$ better reflect the pattern of compounded, long term impact of both rill and sheet erosion and averaging over a long term sequence of large and small events. Caution should be used when interpreting the results from USPED, because the RUSLE parameters were developed for simple plane fields and detachment limited erosion.

We have already computed slope, aspect, and flow accumulation map layers. We now need to use them to compute the estimate of sediment flow and its divergence, while taking into account variability in land cover.

First, we create a C factor map layer by recoding the land cover map, where we will recode the forest to 0.0005, grass to 0.005, agricultural fields, including the grapes and gravel road to 0.5, dirt road to 0.7, and bare soil to 0.8:

```
r.report lw_landcov.type
r.recode lw_landcov.type out=cf1.6
> 1:1:0.
> 2:2:0.0005
> 3:3:0.
> 4:4:0.8
> 5:5:0
> 6:6:0.5
> 7:7:0.5
> 8:8:0.5
> 9:9:0.
> 10:10:0.7
> 11:11:0.005
> end
```

```
r.colors cf1.6 col=rules
> 0 aqua
> 0.0005 0 150 0
> 0.005 green
> 0.5 orange
> 1. brown
> end

d.rast cf1.6
```

To account for variability in soils and to incorporate the relative impact of rainfall, we also include the K and R = 270 factors. We use the topographic sediment flow map `sedflow.6` (m = n = 1) and compute the net erosion/deposition using the equation 12.4:

```
r.mapcalc "qsx=270*@lw_soil.kfact*cf1.6*sedflow.6*cos(as.6t10)"
r.mapcalc "qsy=270*@lw_soil.kfact*cf1.6*sedflow.6*sin(as.6t10)"
r.slope.aspect qsx dx=qsx.dx
r.slope.aspect qsy dy=qsy.dy
r.mapcalc "erdep.6=qsx.dx + qsy.dy"
r.colors erdep.6 col=rules
 fp: Data range is -13803.640 to 17051.835
> -15000 100 0 100   #dark magenta
> -100 magenta
> -10 red
> -1 orange
> -0.1 yellow
> 0 200 255 200      #light green
> 0.1 cyan
> 1 aqua
> 10 blue
> 100 0 0 100        #dark blue
> 18000 black
> end

d.rast erdep.6
```

Red-orange-yellow areas show erosion and blue shades represent deposition. Obviously, concentrated flow appears to be the largest source of sediment pollution which we have addressed in the previous paragraph. About one magnitude lower, but still very high erosion is predicted in the agricultural fields, if they were managed as a single, homogeneous area which can be bare for several weeks or even months. We can also see that not all of the eroded soil will be transported out of the fields. A substantial portion can be deposited directly in the field concave areas and an additional amount is deposited on the border of the field, where water is slowed down by grass. To improve the stability of the fields, they have been divided into subsections which are planted with

different crops. You can use the imported land cover map with divided fields to experiment with different crops and vegetation to find which approaches work the best.

An alternative approach would be to base the new land use plan on the erosion risk map. You can create the high erosion risk map using `r.recode` (net erosion/deposition D > 10, 20, 50) and assign those areas high density vegetation (*C* factor 0.001), while keeping the same cover everywhere else:

```
r.recode erdep out=erosion.hot
> -3000:-50:1
> -50:-20:2
> -20:-10:3
> -10:3000:0
> end

d.rast erosion.hot
r.mapcalc "cf2.6=if(erosion.hot,0.001,cf1.6)"
d.rast cf2.6
```

Now we recompute the USPED model using the above described combination of map algebra and `r.slope.aspect`. After displaying the resulting map, we can see that the range of erosion and deposition is much lower. Some areas which had erosion still have it, but at a smaller rate, and some areas became depositional. We can compare the land use composition by creating labeled ranges for *C* factor maps using `r.support` and then running `r.report` for both *C* factor map layers.

External models can be used for more complex modeling of pollutant transport, usually involving solution of partial differential equations (Mitasova et al., 2002), see for example *Path sampling modeling*[8], coupled with GRASS. However, GRASS can be relatively easily linked to any model using libgrass[9] or import and export of input data and results (Mitasova et al., 2001).

More complex, dynamic study of watershed hydrology in terms of predicting surface and subsurface flow and related phenomena can be performed using several current and older versions of hydrologic models integrated with GRASS, such as `r.topmodel`, `r.water.fea` and `r.hydro.CASC2D`. Use of these models requires some hydrologic background, especially familiarity with hydrologic terminology and access to input data which are not as widely available as basic GIS map layers. Use of these models is beyond the scope of this book; however, it is important to note that to fully evaluate the impact of spatial distribution of land use, this type of model is needed. They capture such important effects as the reduced velocity of water flow and higher infiltration in areas covered by dense vegetation, or increased risk of flooding due to development when vegetated area is replaced by an impervious surface, for example by a parking lot. Hydrologic modeling is also an important component of several non-point source pollution models which were linked to GRASS, such as

AGNPS and ANSWERS. Unfortunately, GRASS versions of these models are no longer supported, except for SWAT.[10]

NOTES

1 RUSLE documents, `http://www.sedlab.olemiss.edu/rusle/`
2 RUSLE for ArcView,
`http://abe.www.ecn.purdue.edu/~engelb/agen526/gisrusle/gisrusle.html`
3 Wake county GIS site
`http://www.wakegov.com/county/propertyandmapping/gisdigitaldata.htm`
4 Lake Wheeler Map Section
`http://lnweb02.co.wake.nc.us/gis/gismaps.nsf/0762-1619!OpenPage`
5 GRASS Tutorials related Web site,
`http://mpa.itc.it/grasstutor/`
6 North Carolina Flood Mapping Program,
`http://www.ncfloodmaps.com/`
7 National MUIR schema,
`http://www.statlab.iastate.edu/soils/muir/schema_nat.html`
8 Path sampling modeling,
`http://skagit.meas.ncsu.edu/~helena/publwork/Gisc00/astart.html`
9 libgrass software,
`http://grass.itc.it/related_projects.html`
10 SWAT software, `ftp://brcsun0.tamu.edu/pub/swat/`

Chapter 13

USING GRASS WITH OTHER OPEN SOURCE TOOLS (⇑)

GRASS is one of many Free Software projects in the GIS world, however, it is the only full featured free GIS at time. A comprehensive list of more than hundred free GIS projects is available online at the FreeGIS Project Web site.[1] The use, development and support of Free GIS Software is promoted at this site, as well as the use and release of publicly available geographic data. Some free GIS projects can provide additional functionality to GRASS by addressing some of its unsolved or intentional constraints.

Within this chapter, we first highlight procedures that extend the geostatistical analysis capabilities of GRASS. We focus on two statistics software packages, the `gstat` and the `R` project. This chapter does not try to cover the theory of geostatistics. Excellent other books on theory and applications are available, such as Cressie, 1993, Bailey and Gatrell, 1995, and Webster and Oliver, 2001. In relation to the `R` program it is useful to read Chambers and Hastie, 1992, Venables and Ripley, 2000, as well as Venables and Ripley, 2002.

After a brief look at GPS related software tools we close this chapter with a demonstration of fast Web mapping through `UMN/MapServer` linked directly to GRASS for reading GIS data from a GRASS LOCATION.

Maas river bank soil pollution data. Throughout the following sections we use the Maas river bank soil pollution data (Limburg, The Netherlands, Burrough and McDonnell, 1998). These data are provided in the `gstat` package and used in examples in its manual. This data set is implemented also in the R/GRASS interface package. The Maas river bank soil pollution data are sampled along the Dutch bank of the river Maas (Meuse) north of Maastricht. This is a flood plain of the river Maas, not far from where the Maas enters the Netherlands (Borgharen, Itteren, about 3 to 5 km north of Maastricht).[2] The river Maas is at the north-west border of the project area, traversing the area in

north-east direction. Burrough and McDonnell, 1998, use a subset of the same data set in their book (reduced area). The data set provided with the R/GRASS interface was re-projected from the Dutch standard coordinate system (TDN coordinates in stereographic projection) to UTM coordinate system zone 32, on WGS84 ellipsoid. A GRASS LOCATION was defined with following parameters: projection UTM, ellipsoid WGS84, zone 32, north 5652930, south 5650610, west 269870, east 272460, nsres 10, ewres 10, rows 232, cols 259. A pre-defined LOCATION including the data stored column-wise in sites lists can be downloaded from the GRASS Web site.[3] The data sets are stored as sites lists plus one raster map; they can be used to experiment with interpolation or other methods.

This data set contains the following columns (topsoil data were collected as bulk samples during fieldwork in 1990 within a radius of 5 m according to Burrough and McDonnell, 1998:102, 309):

East, north (UTM zone 32 coordinates in meters); x, y (local coordinates in meters); elev (elevation above local reference level in meters); d.river (distance from main river Maas channel in meters); Cd (cadmium in ppm); Cu (copper in ppm); Pb (lead in ppm); Zn (zinc in ppm); LOI (percentage organic matter loss on ignition); flfd (flood frequency class, 1: annual, 2: 2-5 years, 3: every 5 years); soil (3 unnamed soil types).

13.1. GEOSTATISTICS WITH GRASS AND GSTAT

The `gstat`[4] package is Free Software for geostatistical modeling, prediction and simulation in one, two or three dimensions (Pebesma and Wesseling, 1998 and Pebesma, 2001). It requires the `gnuplot`[5] graphical plotting software for the display of empirical variograms and variogram models.

With `gstat` you can perform geostatistical modeling in terms of generating empirical (sample) variograms and cross variograms (or covariograms). The software calculates sample (co-)variograms from ordinary, weighted or generalized least squares residuals. Models can be fitted to these variograms to predict data distributions. Using weighted least squares, nested models are fitted to sample (co-)variograms. Restricted maximum likelihood estimation of partial sills is also implemented. Variograms are plotted using the plotting program `gnuplot`, when working in interactive variogram modeling user interface.

The `gstat` software provides prediction and estimation using a model that is the sum of a trend modeled as a linear function of polynomials of the coordinates or of user-defined base functions, and an independent or dependent, geostatistically modeled residual. This allows for simple, ordinary and univer-

sal kriging, simple, ordinary and universal cokriging, standardized cokriging, kriging with external drift, block kriging and "kriging the trend", as well as uncorrelated, ordinary or weighted least squares regression prediction. Simulation in `gstat` comprises uni- or multivariable conditional or unconditional multi-Gaussian sequential simulation of point values or block averages, or (multi-) indicator sequential simulation (features cited after Pebesma, 2001).

The gstat/GRASS interface allows the user to read point data from site lists and raster maps. This requires to have the GRASS support compiled into `gstat`. You can check your version with flag `-v`:

```
gstat -v
```

The line "with libraries" must list "grass" besides other supported formats (e.g. "grass gdal netcdf").

Output of `gstat` (prediction or simulation results) is written to raster maps and also to site lists. You need to run `gstat` from inside GRASS as the program requires the GRASS environment to internally set up the LOCATION definitions. When a subregion is set in GRASS, `gstat` will only interpolate or simulate the raster cells according to the current region. The variables of interest need to be floating point numbers (DOUBLE) in sites list or stored in a raster map. The instructions for `gstat` are stored in an ASCII file. When using GRASS sites lists as input maps, following column order conventions have to be followed:

```
Easting|Northing|#site_no %FP_data [%FP_data]
```

These are the same conventions as for standard GRASS sites lists. The program `gstat` reads GRASS site data from the current MAPSET with the `data()` function. Variable positions are defined as:

```
x=1: coordinate column 1 contains the x-coordinate
y=2: coordinate column 2 contains the y-coordinate
z=3: coordinate column 3 contains the z-coordinate (optional)
v=1: data column 1 contains the first data (measurement)
          variable, when 0, a grid map is read
```

To illustrate how it works, we run a sample session based on the "Maas river bank" data set. First start GRASS with the Maas UTM LOCATION, then copy the `Zn` (zinc) concentrations sites map to the current MAPSET:

```
grass53 /usr/local/share/grassdata/maas/user1/
g.copy sites=Zn,zinc
s.info zinc
```

The following example is based on the manual of `gstat`[6]. Store the following commands to the file `gstat.maas1.zn` in your home-directory:

```
# Two variables with (initial estimates of) variograms,
# start the variogram modeling user interface
data(zinc): 'Zn', x=1, y=2, v=1;
data(ln_zinc): 'Zn', x=1, y=2, v=1, log;
variogram(zinc): 10000 Nug() + 140000 Sph(800);
variogram(ln_zinc): 1 Nug() + 1 Sph(800);
```

As the zinc concentrations are stored as first DOUBLE attribute in the sites list (in ppm, reported by `s.info zinc`), we select this data column through `v=1`. Run the analysis by:

```
gstat gstat.maas1.zn
```

The program starts to analyze the data and subsequently displays univariate statistics:

```
gstat: Linux version 2.4.3 (04 January 2004)
Copyright (C) 1992, 2004 Edzer J. Pebesma
using Marsaglia's random number generator
data(zinc): gisrc: [/home/neteler/.grassrc5]
GRASS site list zinc: 0 cat, 2 dim, 0 str, 1 dbl.
gstat/grass: 98 sites read successfully.
             zinc     (GRASS site list)
attribute: col[1]     [x:] x_1     : [    269800,    272500]
n:            98      [y:] y_2     : [5.6506e+06, 5.653e+06]
sample mean: 481.031  sample std.:        398.808
GRASS site list zinc: 0 cat, 2 dim, 0 str, 1 dbl.
data(ln_zinc): gstat/grass: 98 sites read successfully.
          zinc     (GRASS site list)
attribute: log(col[1])  [x:] x_1    : [    269800,    272500]
n:           98     [y:] y_2       : [5.6506e+06, 5.653e+06]
sample mean:  5.87065 sample std.:          0.778309
[starting interactive mode]
press return to continue...
```

After pressing <ENTER> we reach the main menu, which allows us to interactively analyze the loaded data set:

```
            gstat 2.4.3  (04 January 2004), gstat.maas1.zn

 enter/modify data
 choose variable    : zinc
 calculate what     : semivariogram
 cutoff, width      : 1204.15, 80.2765
 direction          : total
 variogram model    : 10000 Nug(0) + 140000 Sph(800)
 fit method         : no fit
>show plot <Tab>
[...]
Command: _
```

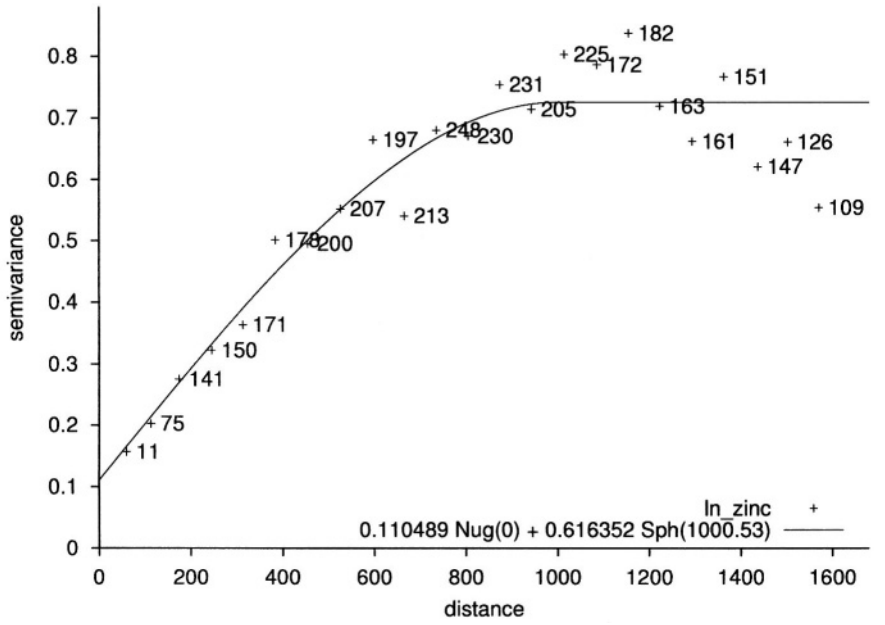


Figure 13.1. gstat/GRASS: Semivariogram of zinc contaminations of the Maas river bank soil samples (variogram model: WLS, weights n(h))

After reaching the menu you can move around with cursor keys. Now choose the variable `ln_zinc` (logarithmic transformed zinc concentrations) with <ENTER>. Set the `cutoff` (lag distance) to 1600 and `width` to 70. Then select for `fit methods` "WLS, weights n(h)" (WLS is weighted least squares, other methods are also available). Now the variogram model will be fitted when hitting the <TAB> key or selecting `show plot <Tab>`. The resulting semivariogram is shown in Figure 13.1.

Zinc contamination kriging example. In the next example (adapted from Pebesma, 2001:12) we will include a raster MASK and perform ordinary kriging prediction from the zinc data. This will result in a raster surface map with predicted distributed zinc contaminations and the predicted kriging error. The `gstat` instructions file looks as follows (store it as file `gstat.maas2.zn`):

```
# ordinary kriging prediction
#
data(zinc): 'zinc', x=1, y=2, v=1;
variogram(zinc): 0.0717 Nug(0) + 0.564 Sph(917.8);
mask: 'maasmask';
```

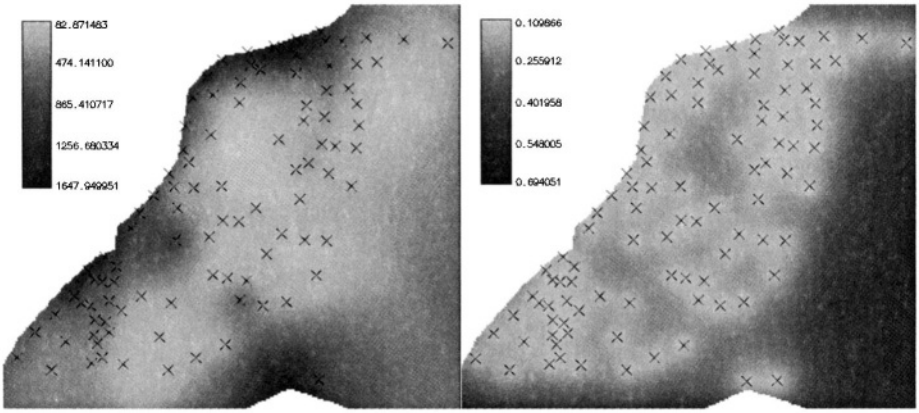


Figure 13.2. gstat/GRASS: Ordinary kriging prediction of zinc contaminations on the Maas river bank. Left: predicted zinc contaminations [ppm], right: prediction error

```
predictions(zinc): 'zinc_pr.map';
variances(zinc): 'zinc_var.map';
```

In general, output grid maps are always written in the same format as the input mask map. Besides the GRASS format, `gstat` also supports other GIS formats. Before running the prediction, we have to generate the GRASS raster map `maasmask`. The Maas LOCATION contains a raster map `maasbank` which covers the river bank area. We generate the desired binary map from this map, then we run the kriging prediction:

```
r.mapcalc "maasmask=if(maasbank)"
d.rast maasmask
d.sites zinc
gstat gstat.maas2.zn
```

During calculations the program will report similar user messages:

```
gstat: Linux version 2.4.3 (04 January 2004)
Copyright (C) 1992, 2004 Edzer J. Pebesma
using Marsaglia's random number generator
data(zinc): gisrc: [/home/neteler/.grassrc5]
GRASS site list zinc: 0 cat, 2 dim, 0 str, 1 dbl.
gstat/grass: 98 sites read successfully.
             zinc      (GRASS site list)
attribute: col[1]      [x:] x_1    : [    269870,    272460]
n:             98      [y:] y_2    : [5.65061e+06,5.65293e+06]
sample mean: 481.031  sample std.:          398.808
[using ordinary kriging]
ncols 259
nrows 232
initializing maps ..
```

```
CREATING SUPPORT FILES FOR zinc_pr.map
CREATING SUPPORT FILES FOR zinc_var.map
100% done
```

Two new raster maps have been generated, which contain the predicted zinc distribution (`zinc_pr.map`) and the distributed error (`zinc_var.map`). Using `d.frame` we can display both maps side-by-side in the GRASS monitor:

```
r.colors zinc_pr.map col=gyr
r.colors zinc_var.map col=gyr
d.frame -e
d.frame -c at="0,100,50,100"
d.frame -c at="0,100,0,50"
d.rast zinc_pr.map bg=white
d.sites zinc col=black
d.legend -ms zinc_pr.map col=black
d.frame -s
d.rast zinc_var.map bg=white
d.legend -ms zinc_var.map col=black
d.sites zinc col=black
```

The result is shown in Figure 13.2. For further details and examples please refer to the `gstat` documents.

13.2. SPATIAL DATA ANALYSIS WITH GRASS AND R

The "R data analysis programming language and environment" (Ihaka and Gentleman, 1996, available from Internet[7]), a dialect of S (Becker, 1988), is an extensible system which can be connected directly to GRASS. `R` consists of a base package and extensions that can be downloaded from the project Web site. A regular newsletter informs about recent changes and improvements.[8] Besides classical methods, graphical and modern statistical techniques are implemented in the base `R` library and supplementary packages. Latter comprise packages for point pattern analysis, geostatistics, exploratory spatial data analysis and spatial econometrics. While `R` is a general data analysis environment, it has been extensively used for modeling and simulation. Through the R/GRASS interface (Bivand, 2000, Bivand and Gebhardt, 2000, Bivand and Neteler, 2000, Furlanello et al., 2003) the geospatial analysis capabilities of GRASS are substantially improved.

For the integration of `R` into GRASS you need to run `R` from the GRASS shell environment. The interface dynamically loads compiled GIS library functions into the `R` executable environment. The GRASS metadata about the LOCATION'S regional extent and raster resolution are transferred to `R`. In interpreted mode, when the interface installation was not compiled with the GRASS GIS library, this is taken from `g.region`. The R/GRASS interface

– like GRASS modules in general – assumes that the user needs the current resolution, not the initial resolution of the map layer. The current interface is supporting raster and site data. Work on an interface for vector data is ongoing.

Besides the base package of R it is useful to install also the following contributed extensions: akima, fields, geoR, grid, lattice, MASS, scatterplot3d, spatial, and stepfun (available from the R Web site). Additional packages in terms of spatial-temporal analysis are focused on autocorrelation, spatial point patterns, time series or wavelets.

Note that in this section we omit the \ character for long lines as it is not allowed in R. Long, broken lines are indicated by the indent in the next line.

Installation of the R/GRASS interface. The installation of the R/GRASS interface[9] is very easy and can be done by a single command (you probably have to be user "root" for this).

If your computer is connected to the Internet, you can install packages within an R session. Start R and launch the command:

```
install.packages("GRASS")
```

This will download the latest version of the selected package, unpack, compile and install it. From time to time the installed extensions should be checked for updates. The following command will download the R package list, compare to the local installation and upgrade installed packages if new versions are available:

```
update.packages()
```

Offline, you can install extra packages on command line:

```
R CMD INSTALL GRASS_0.2-13.tar.gz
```

As the interface is maturing, the version number is subject to change. Also the path to R depends on your local installation. To start a quick exploration of standard R, simply run:

```
R
demo()
demo(graphics)
q()
```

The function q() finishes a R session. When leaving R with q() you will be asked: "Save workspace image? [y/n/c]:". If answering y, the objects are stored within the local directory into the hidden file .RData. When launching R next time in this directory, the objects will be read into the system and you can continue with your work.

As mentioned above, contributed packages extend immensely the functionality of GRASS in conjunction with the R/GRASS interface. To find out which packages are already available on your system, type within R:

```
library()
```

You can load an installed package by entering its name as a parameter for this function. Some packages also provide examples for their functions. We try an example for fitting a trend surface, the function is provided by library spatial:

```
library(spatial)
example(surf.ls)
?surf.ls
```

The ? command displays the function's help text (leave with q). Help pages are also stored in HTML format, you can open a HTML browser with:

```
help.start()
```

The pages provide package explanations and a local search engine.

When using R in batch mode you can develop GRASS scripts, which are directly utilizing R functionality in GRASS user environment. In Section 13.2.3 we show sample scripts.

13.2.1 Spearfish data set analysis

To illustrate how to apply R to your data, we present several examples based on the Spearfish data set. As an exercise, we can generate a trend surface of the distributed pH value within the Spearfish region. After starting GRASS with Spearfish LOCATION and resetting the region to default settings, run R within GRASS:

```
grass53 /usr/local/share/grassdata/spearfish/user1
g.region -dp
R
```

Within R the R/GRASS interface is loaded as follows:

```
library(GRASS)
G <- gmeta()

#show loaded environment:
class(G)
summary(G)
```

By this, we have loaded the library of interface functions and initialized an object named G that includes metadata about the location. This object, like

many in R, has a class grassmeta- which is used for selecting functions that are applied to the object, for example, summary() or plot(). The class of the object, which is the first argument to a generic function, passes the object on to a function specifically for that object class, so function summary(G) gives a summary appropriate for objects of class grassmeta. A little later, we will see summary() used on numeric data, and the plot() function used with G as the first argument.

Now we are ready to perform geospatial analysis of GRASS raster and site data. As a prerequisite to generate the trend map of pH values with R, the pH values map is read into R environment (with c(F) category import set to False):

```
soilsph <- rast.get(G, "soils.ph", c(F))
str(soilsph)
names(soilsph) <- c("ph")
str(soilsph)
```

The function rast.get() reads the map soils.ph from GRASS and stores it into the R object soilsph. The str() function shows a brief view of the structure of the soilsph object. It starts with "NA" (no-data) values because the GRASS soils.ph map contains NULL values at the borders. For later convenience the imported variable is renamed from soilsph$soils.ph to soilsph$ph with the names() function.

Univariate statistics (minimum, maximum, 1st and 3rd quartile, median, mean and the number of NA's) of the pH values can be displayed with summary() function. Up to now the recently imported map stored in data object soilsph contains only the data values, but not the related coordinates. To complete it, we generate a so-called data frame which allows us to store associated variables in one object. In our case, the data frame will carry east and north coordinates and the z values, which represent the ph values:

```
summary(soilsph$ph)
soils.ph.frame <- data.frame(east(G), north(G), soilsph$ph)
str(soils.ph.frame)
system("r.report soils.ph")
```

Since the original soils.ph map contains cells with "no soil data" label (check with r.report) we have to set these values to NA. For further processing we rename the variables to standard naming convention. Most functions expect the variable names to be "x", "y" and "z". Note that when calling an object in R by name, the object contents will be displayed (e.g. all cell values):

```
soilsph$ph[soilsph$ph == 0] <- NA
names(soils.ph.frame) <- c("x", "y", "z")
summary(soils.ph.frame)
soils.ph.frame$z
```

To see all data objects which are currently loaded into R, use the `ls()` function. Next we want to plot the map stored in object `soils.ph.frame`:

```
ls()
plot(G, soils.ph.frame$z)
```

A new window opens with the pH values map displayed. The map colors may differ from GRASS display, but they can be adjusted. The object `soils.ph.frame` is now prepared for further analysis. Next we can calculate the cubic trend surface with function `surf.ls()` (Venables and Ripley, 2002). The function does not accept no-data (NA) values, so we have to mask all NAs by `na.omit()` function. The functions to fit the trend surface through the data points are provided by `spatial` package which we have to load first:

```
library(spatial)
ph.ctrend <- surf.ls(3, na.omit(soils.ph.frame))
summary(ph.ctrend)
```

This function fits a polynomial model to the data set. The parameters for the trend surface modeling function are the degree of polynomial surface (in our case it is 3 for a cubic polynomial) and the data frame with NAs omitted. The calculations require some memory and computational power. The generated trend surface model is then stored into the object `ph.ctrend`. From this model the trend surface map is evaluated by function `trmat.G()`. The resulting map can be visualized either as contour lines (function `contour.G()`) or as solid surface (`plot()` function). To shorten commands, we evaluate the trend surface on the fly directly within the `plot()` function which draws the surface into the R graphics window:

```
plot(G, trmat.G(G, ph.ctrend), col=terrain.colors(20))
```

Another option is to store the trend surface map into its own object and plot it later (as a solid surface or labeled contour lines):

```
trendSurf <- trmat.G(G, ph.ctrend)
plot(G, trendSurf, col=terrain.colors(20))
contour.G(G, trendSurf)
title("Cubic trend surface of pH values in Spearfish region")
```

It is possible to redirect the R graphical output into a file. For example, to export plots into a Postscript file (function `postscript()`), use:

```
postscript("trendSurf.ps")
plot(G, trendSurf, col=terrain.colors(20))
contour.G(G, trendSurf, add=T)
title("Cubic trend surface of pH values in Spearfish region")
dev.off()
```

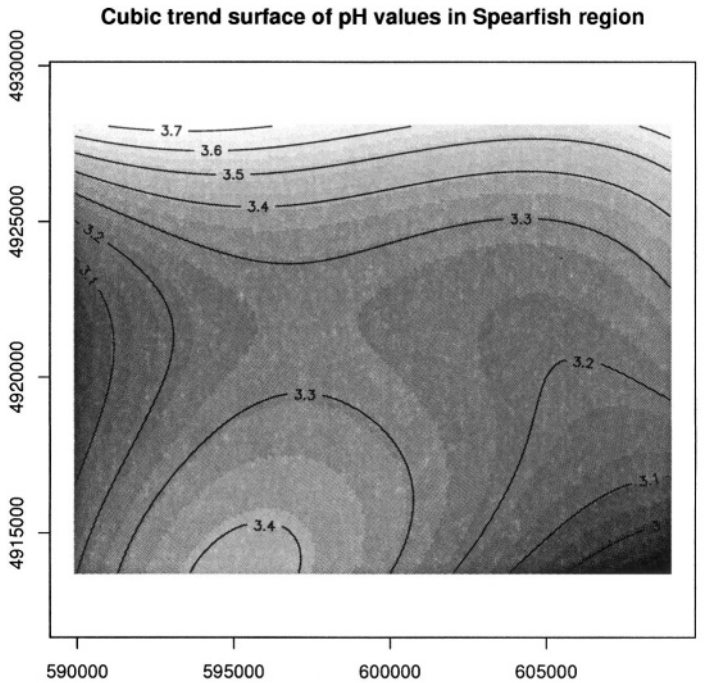


Figure 13.3. R/GRASS: Cubic trend surface of pH values in Spearfish region (calculated at 100 m raster resolution)

The additional parameter `add=T` for the function `contour.G()` overlays the contour lines over the previous map plot. The `dev.off()` call writes the requested postscript file. Other devices are `pdf()`, `pictex()`, `x11()` (the screen), `png()`, `xfig()` and `jpeg()`. To generate EPS (Encapsulated PostScript) files compatible for integration into other documents, e.g. into LaTeX, you may want to use the function `dev.copy2eps()`. The cubic trend surface from Spearfish pH values is shown in Figure 13.3.

Sometimes it may be necessary to free some memory, so we want to remove objects:

```
ls()
rm(ph.ctrend,soilsph,soils.ph.frame,trendSurf)
history()
q()
```

Note that removing objects before leaving the `R` program is not needed (we just wanted to show how to do it in general). The function `history()` works similarly as the UNIX `history` command: It displays all commands used in the current `R` session.

Working with reclassified raster maps. In this sample session we want to find the average elevation of each of the Spearfish soil classes and the area occupied by each of the classes in hectares. In GRASS, this is done by creating a reclassed map layer in `r.average`, which assigns mean values to the category labels, and `r.report` to access the results:

```
g.region -p rast=elevation.dem
r.average base=soils cover=elevation.dem output=avheight
r.report soils units=h
r.report avheight units=h
```

Remember that documentation material, such as soil names explanations for the Spearfish soils map, can be found on the GRASS Web site ("sample data" section).

To answer the same question in `R` using the R/GRASS interface, we import the two required maps into `R`, `soils` as factor map (classes), and `elevation.dem` as a numeric vector:

```
g.region -p rast=elevation.dem
R
library(GRASS)
G <- gmeta()
spearfish <- rast.get(G, c("soils", "elevation.dem"), c(T, F))
names(spearfish) <- c("soils.f", "dem")
str(spearfish)
```

While the soil classes are imported with category label support, the labels for the elevation are omitted. Again we rename the variables for convenience. `R` provides the `tapply()` function which lets us apply the declared function `mean()` to the subsets of the first argument grouped by the second, giving the results we need. The count of cells by land use type are given by `table()`, which we use to convert square meters to hectares using the metadata on cell resolution. Finally, as an advantage over GRASS functionality, we can construct a boxplot of elevation by soil unit, making box width proportional to the number of cells in each land use class using the table `soilareas` (note that we drop the areas filled with NA values by subsetting to the row range 2 - 55 with `soilareas[2:55]`). We get the number of rows, number the of soil types (55 including the no-data area) from the `dim()` function which returns number of rows and columns:

```
soil.h.mean <- tapply(spearfish$dem, spearfish$soils.f, mean)
soil.h.min <- tapply(spearfish$dem, spearfish$soils.f, min)
soil.h.max <- tapply(spearfish$dem, spearfish$soils.f, max)
soil.h.range <- tapply(spearfish$dem, spearfish$soils.f, range)
soilareas <- table(spearfish$soils.f)*((G$ewres*G$nsres)/10000)
soiltable <- cbind(soil.h.mean,soilareas,soil.h.min,soil.h.max)
```

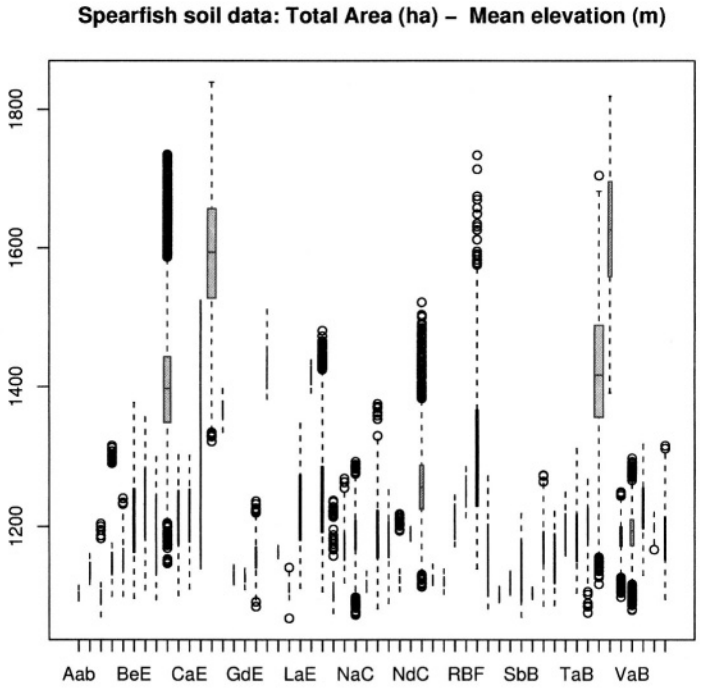


Figure 13.4. R/GRASS: Boxplot of soil type distribution against altitude in Spearfish region (calculated at 100 m raster resolution). The box sizes are representing the area sizes

```
colnames(soiltable) <-c("Average Height (m)","Tot. Area (ha)",
                        "Min. Height (m)", "Max. Height (m)")
soilareas
dim(soilareas)
boxplot(spearfish$dem ~ spearfish$soils.f,
        width=soilareas[2:55], col="gray", horizontal=TRUE)
title("Spearfish soil data: Total Area (ha) -
                        Mean elevation (m)")
soiltable
soil.h.range
```

The boxplot (see Figure 13.4) does not display all soil names due to the space limitations. When directly calling object `soiltable` in the terminal window the values table is printed into the terminal. In `soil.h.range` the elevation ranges per soil name are stored. While some soil types are found in a wider elevation range (e.g. VBF, Vanocker-Citadel association, from 1117 m - 1705 m altitude), others are restricted to much narrower range (e.g. NaC, Nevee silt loam, from 1072 m - 1292 m altitude). Boxplots such as above help to show more about the empirical distributions of cell values.

Analyzing interpolated maps. The empirical cumulative distribution function (ECDF) can be useful for identification of a bias towards contours in interpolated DEMs. It is also known as hypsometric integral. After generating a data frame of the input raster map, the ECDF function can be plotted directly. To improve legibility, we use a `for()` loop to draw lines in the range of the elevation. After starting `R` and loading the R/GRASS interface, we run:

```
library(stepfun)
elev <- rast.get(G, "elevation.dem")
elev.f <- data.frame(east(G), north(G), elev$elevation.dem)
names(elev.f) <- c("x", "y", "z")
summary(elev.f)
plot(ecdf(elev.f$z), verticals=T, do.points=F,
                    xlab="elevation", main=NULL)
for (i in seq(1000,1900,20)) lines(c(i,i), c(0,1), lty=2,
                                                col="red2")
title("Spearfish elevation.dem data: ECDF plot")
```

As shown in Figure 13.5 the plot looks very smooth. This indicates that the Spearfish raster map `elevation.dem` does not have waves along contours.

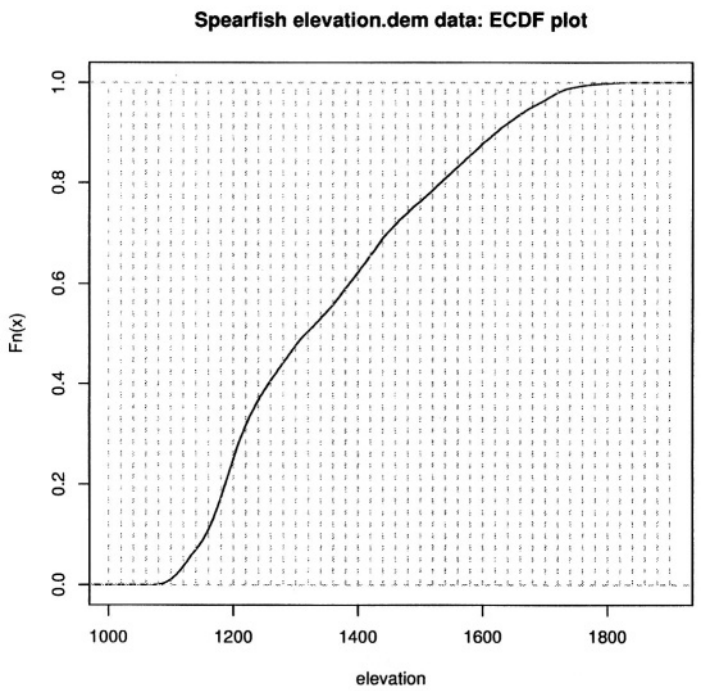


Figure 13.5. R/GRASS: Empirical cumulative distribution function (ECDF) plot to identify interpolation artifacts in Spearfish integer `elevation.dem` map

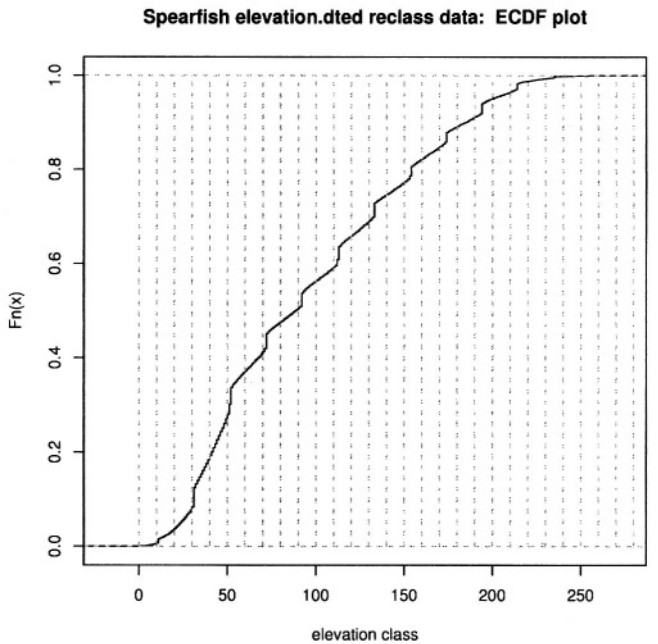


Figure 13.6. R/GRASS: Empirical cumulative distribution function (ECDF) plot to identify interpolation artifacts in Spearfish reclassified elevation.dted map

For comparison we can do the same analysis with the `elevation.dted`, a map which was reclassified from the original elevation data:

```
dted <- rast.get(G, "elevation.dted")
dted.f <- data.frame(east(G), north(G), dted$elevation.dted)
names(dted.f) <- c("x", "y", "z")
plot(ecdf(dted.f$z), verticals=T, do.points=F,
             xlab="elevation class", main=NULL)
for (i in seq(0,300,10)) lines(c(i,i), c(0,1), lty=2,
                                        col="red2")
title("Spearfish elevation.dted reclass data:  ECDF plot")
```

As expected, "stairs" are visible in the ECDF plot (see Figure 13.6). They are the result of the reclassification of the original elevation map to a reclassed map.

An additional useful tool for analysis of interpolation artifacts is the density plot over elevation (smoothed histogram). The function `density()` computes kernel density estimates with the given kernel (default: Gaussian) and given bandwidth. If the DEM was interpolated from contour lines, we can probably

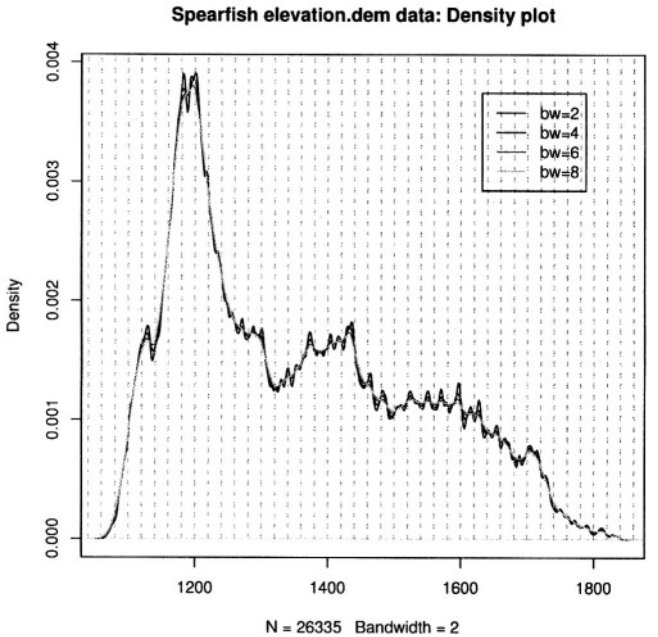


Figure 13.7. R/GRASS: Density plot of Spearfish `elevation.dem` data

see a local over-representation of the contours compared to the interpolated values (depending on interpolation method, see Jacoby, 1997:23pp for details).

Note that NA values are not permitted in the `R density` function, therefore we mask them on the fly. The legend has to be placed by mouse into the graph, the `locator()` function silently waits for a click of the middle mouse button:

```
plot(density(na.omit(elev.f$z), bw=2), col="black",
  main="Spearfish elevation.dem data: Density plot")
lines(density(na.omit(elev.f$z), bw=4), col="green")
lines(density(na.omit(elev.f$z), bw=6), col="brown")
lines(density(na.omit(elev.f$z), bw=8), col="blue")
for (i in seq(1000,1900,20)) lines(c(i,i), c(0,0.015), lty=2,
                                                col="grey")
legend(locator(), legend=c("bw=2", "bw=4", "bw=6", "bw=8"),
          col=c("black", "green", "brown", "blue"), lty=1)
```

Instead of the `locator()` function you can also specify x,y coordinates related to the current coordinates' ranges. In our example, you may specify `1600, 0.004` instead of `locator()`. The graph (see Figure 13.7) shows only slight artifacts up to a kernel bandwidth of 4 meters. These methods may

be interesting to compare results from different interpolation methods as described in Section 6.4.3 or to verify unknown interpolated data.

13.2.2 Maas river bank soils data analysis

When analyzing spatial data taken at various sample locations, we may be interested in predicting values in unsampled areas. With variogram modeling and kriging we can generate raster surfaces from point spatial data. GRASS does not include functional modules for analysis of variograms, but R provides extended capabilities. See Section 7.3.8 for a short discussion on geostatistics, kriging and splines. The next examples cover basic data analysis while variogram analysis and kriging are not presented due to the complexity of the topic. The R libraries `fields` and `geoR`, partly also the R/GRASS interface provide the relevant functionality and examples.

To do the following examples we must run R from inside GRASS. This time we will use the data set which is stored in the R/GRASS interface (and which was the base for the GRASS Maas LOCATION).

Maas soil data analysis. To set the environment, start GRASS with the "maas" LOCATION and R within GRASS. Note that you can always use `help(function)` to read a function's help page and `example(function)` to run the examples provided for that function. This will also show examples explained in this section as they are based on the interface documentation. Inside R first load the environment and, in this case only for plotting purposes, the library `geoR` (Ribeiro and Diggle, 1999, Ribeiro and Diggle, 2001, see also online[10]) which provides numerous geostatistical functions:

```
grass53 /usr/local/share/grassdata/maas/user1/
g.region -dp
R
library(GRASS)
G <- gmeta()
library(geoR)
```

Next load the "maas" data set (this time directly the built-in data). The Maas data set contains numerous variables. You can plot all variables against each other (compare Burrough and McDonnell, 1998:112):

```
data(utm.maas)
str(utm.maas)
plot(utm.maas)
plot(utm.maas[1:5])
```

You can see the complete list of variables in the "utm.maas" object with `str(utm.maas)`. The second `plot()` command plots only a subset of all variables. As we want to analyze the zinc contamination, we focus on these

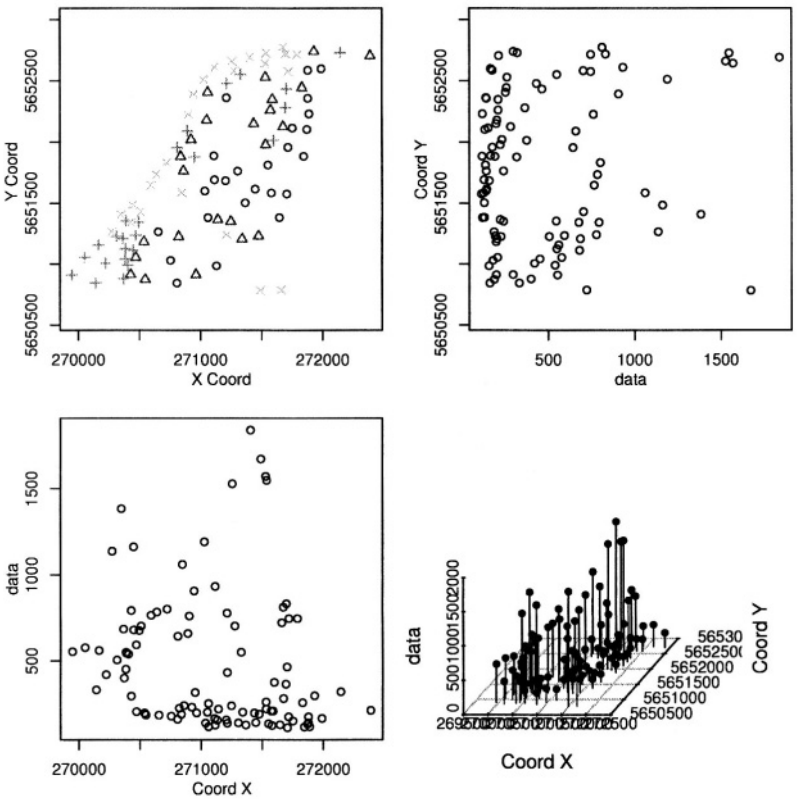


Figure 13.8. R/GRASS: Maas river bank zinc contamination data. Upper left: zinc contaminations quantile plot; upper right: zinc data against y coordinates; lower left: zinc data against x coordinates; lower right: 3D scatterplot

data now. We generate an empty data frame, in which we store the coordinate pairs as matrix and the related zinc (Zn) concentrations in the "data" variable:

```
zinc <- data.frame()
class(zinc) <- "geodata"
zinc$coords <- cbind(utm.maas$east, utm.maas$north)
zinc$data <- utm.maas$Zn
str(zinc)
```

The function `class()` is used to store the class information in the object. The `plot()` function uses this information to invoke the standard data plot as implemented in `geoR` package. As we have compounded several variables into the zinc data object, we can plot numerous graphs right away, according to the definitions in underlying functions of `plot.geodata()`:

```
plot(zinc, scatter3d=T)
quantile(zinc$data)
```

In Figure 13.8 the upper two plots are data locations, the lower two plots show the zinc data against the x and y coordinates. If the R extension scatterplot3d is not installed, the lower right plot will show a normal histogram instead of a 3D point data plot. The quantiles for the upper left plot are generated according to the results of the quantile() function.

The Maas river bank area can be retrieved from the already loaded maasmask data set which is a binary mask for the area. To generate a map with coordinates matching those of the binary mask (which is a R data vector only without coordinates), we export the map to GRASS and re-import it again. This will apply the related coordinates to the mask values:

```
str(maasmask)
rast.put(G, lname="riverbank", maasmask)
riverbank <- rast.get(G, "riverbank")
names(riverbank) <- c("z")
str(riverbank)
plot(G, riverbank$z)
```

Here we are using the plot method for geospatial data as available through the R/GRASS environment (invoked through the G GRASS metadata object).

To plot the individually measured zinc concentrations into the Mass river bank map, we can use the points() and text() functions for setting the labels (pos=4 prints the labels right to the label point, cex=0.75 decreases the label font size):

```
points(utm.maas$east, utm.maas$north, pch=20, col="blue")
text(utm.maas$east, utm.maas$north, utm.maas$Zn,pos=4,cex=0.75)
title("Maas river bank: zinc contamination [ppm]")
```

We see all sampling points labeled with the zinc concentrations measured as parts-per-million (ppm). The river Maas is at the north-west border of the project area, flowing in the north-east direction.

To find out whether the zinc concentration depends from the distance to the river we plot the variables "Zn" against "d.river", both untransformed and transformed using natural logarithm (log() function in R). We can split the output screen into two parts with function par() and display two (or more) plots at the same time (we reset directly after plotting):

```
par(mfrow=c(2,1))
plot(utm.maas$d.river, utm.maas$Zn)
title("Maas river bank: zinc concentrations/distance")
plot(log(utm.maas$d.river), utm.maas$Zn)
title("Maas river bank: zinc concentrations/ln\(distance\)")
par(mfrow=c(1,1))
```

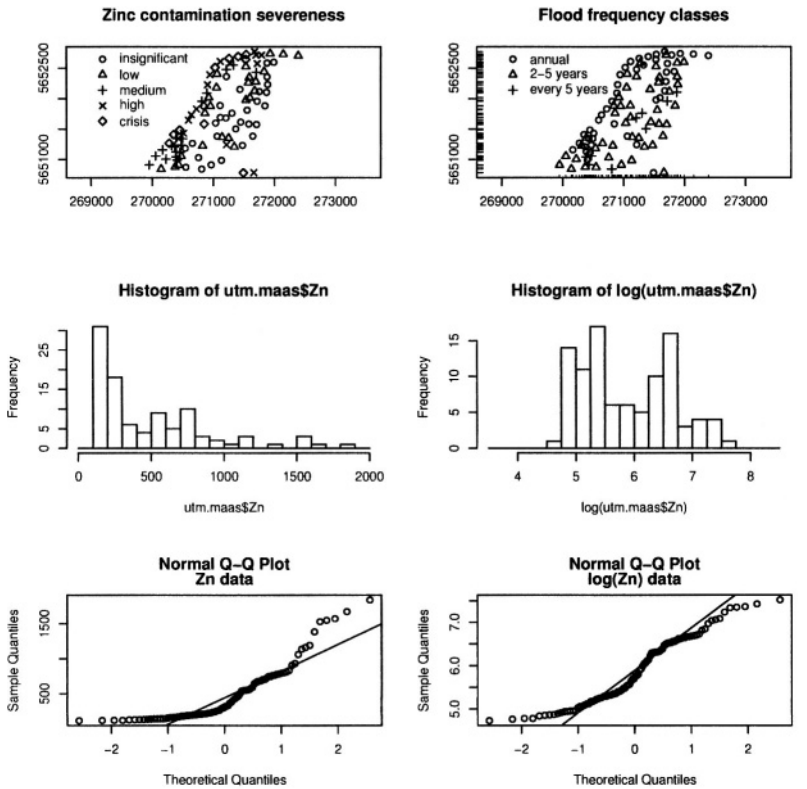


Figure 13.9. R/GRASS: Maas river bank data: zinc contamination analysis. Upper left: classes of zinc contamination severeness; upper right: flood frequency classes: 1=annual, 2=2-5 years, and 3=every 5 years; middle left: histogram of zinc concentrations [ppm]; middle right: histogram of logarithmic transformed zinc concentrations; lower left: QQ plots (Quantile-Quantile plot) of zinc data; lower right: QQ plots of log-transformed zinc data

The plot shows that moving from the river border, the zinc concentrations tend to decrease as expected.

We can look at the zinc data in more detail by analyzing the zinc concentrations for their severeness. First a new object is created with ordered data using the "utm.maas" object which contains the zinc concentrations. We generate five classes with defined thresholds, the example is based on the R/GRASS interface's help pages:

```
Zn.o <- as.ordered(cut(utm.maas$Zn, labels=c("insignificant",
  "low", "medium", "high", "crisis"), breaks=c(100, 200,
  400, 700, 1000, 2000), include.lowest=T))
table(Zn.o)
```

To plot the ordered zinc severeness data as a map with legend and title (compare Burrough and McDonnell, 1998:107), enter:

```
plot(utm.maas$east, utm.maas$north, pch=codes(Zn.o), xlab="",
                                                    ylab="", asp=1)
legend(x=c(270000, 270600), y=c(5652100,5652700), pch=c(1:5),
                                                 legend =levels(Zn.o))
title("Maas river bank: zinc contamination severeness")
```

The `codes()` function is required to access the category numbers for the legend symbols. The legend is placed at certain coordinates into the map. The `levels()` function shows the different category labels for the legend. You can always call a function directly to see which results are generated (e.g. `levels(Zn.o)`). The map is shown in Figure 13.9.

Next we plot a map of flood frequency classes, see Figure 13.9. The function `rug()` allows us to plot small lines at the map border to illustrate the data points position:

```
plot(utm.maas$east, utm.maas$north, pch=utm.maas$Fldf,xlab="",
                                                    ylab="", asp=1)
floodtext <- c("annual","2-5 years","every 5 years")
legend(x=c(270000, 270800), y=c(5652300, 5652700), pch=c(1:3),
                                                 legend=floodtext)
rug(utm.maas$east, side=1, ticksize=0.02)
rug(utm.maas$north, side=2, ticksize=0.02)
title("Maas river bank: Flood frequency classes")
```

We can identify points in this map using mouse (the value will be printed into the map, end the query with right mouse button):

```
identify(utm.maas$east, utm.maas$north, labels=utm.maas$Fldf)
```

Let us have another look at the "flood frequency class Fldf" related to zinc contamination. We can compute the mean and standard deviation (sd) and do the same with log-transformed data (compare Burrough and McDonnell, 1998:107). At the end, we print the results:

```
ZnFldf.mean <- round(tapply(utm.maas$Zn,
      as.factor(utm.maas$Fldf), mean), 2)
ZnFldf.sd <- round(tapply(utm.maas$Zn,
      as.factor(utm.maas$Fldf), sd), 2)
logZnFldf.mean <- round(tapply(log(utm.maas$Zn),
              as.factor(utm.maas$Fldf), mean), 3)
logZnFldf.sd <- round(tapply(log(utm.maas$Zn),
              as.factor(utm.maas$Fldf), sd), 3)
ZnFldf.table <- cbind(ZnFldf.mean, ZnFldf.sd, logZnFldf.mean,
                                                 logZnFldf.sd)
```

```
colnames(ZnFldf.table) <- c("Mean Zn", "SD Zn",
              "  Mean log(Zn)", "  SD log(Zn)")
ZnFldf.table
```

We use the `cbind()` function to paste all calculations into the table `ZnFldf.table`, it looks as follows:

```
  Mean Zn  SD Zn   Mean log(Zn)   SD log(Zn)
1  769.77 423.17          6.484        0.609
2  264.98 176.62          5.421        0.531
3  205.78 105.33          5.239        0.416
```

Given the flood frequency classes as: 1=annual, 2=2-5 years, and 3=every 5 years, it can be seen that high zinc contaminations only appear for areas with annual flooding. Further tests such as t-test analysis (Burrough and McDonnell, 1998:107) of the statistical significance between flood frequency classes 2 and 3 are not covered here. These calculations are described in the R/GRASS interface manual page (enter: `?utm.maas`).

When working with geospatial data, it is always recommended to explore the data distribution. To view the zinc concentrations distribution (using the Zn data and log-transformed Zn data) run:

```
par(mfrow=c(1,2))
hist(utm.maas$Zn, breaks=seq(0, 2000, 100), col="grey")
hist(log(utm.maas$Zn), breaks=seq(3.5, 8.5, 0.25), col="grey")
par(mfrow=c(1,1))
```

The histogram of the non-transformed data shows the typical skewness as usually found for geospatial data. It is normalized through the logarithmical transformation, see Figure 13.9.

Another visual test for normal distribution of data sets are QQ plots (Quantile-Quantile plot):

```
par(mfrow=c(1,2))
qqnorm(utm.maas$Zn)
qqline(utm.maas$Zn)
title("Zn data", line=0.7)
qqnorm(log(utm.maas$Zn))
qqline(log(utm.maas$Zn))
title("log(Zn) data", line=0.7)
par(mfrow=c(1,1))
```

Figure 13.9 depicts both graphs. The QQ plot of the log-transformed data is much closer to normal distribution than the plot of untransformed zinc data.

A method for generating a smooth continuous raster surface from spatially distributed sample points is the "kernel density" plot. It uses a moving Gaussian kernel which represents a bivariate probability density function (Bailey

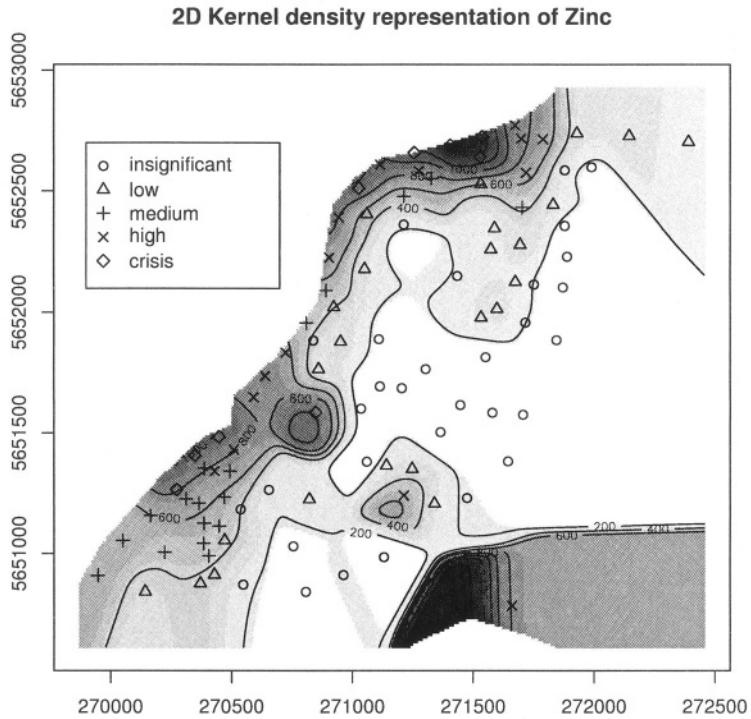


Figure 13.10. R/GRASS: Maas river bank soil data: zinc contamination 2D kernel density (bandwidth: 300m)

and Gatrell, 1995:84-88). We generate a kernel density plot from the zinc data as follows:

```
plot(G, kde2d.G(G, utm.maas$east, utm.maas$north,  h=c(300,300),
                      Z=utm.maas$Zn)*maasmask, col=grey(20:1/20))
contour.G(G, kde2d.G(G, utm.maas$east, utm.maas$north,
         h=c(300,300), Z=utm.maas$Zn)*maasmask, add=T)
points(utm.maas$east, utm.maas$north, pch=codes(Zn.o))
legend(x=c(269900, 270700), y=c(5652100, 5652700), pch=c(1:5),
                                              legend=levels(Zn.o))
title("2D Kernel density representation of zinc")
```

Figure 13.10 shows the resulting map. During calculation we have multiplied the zinc data with a mask `maasmask` to obtain kernel densities only for the project area.

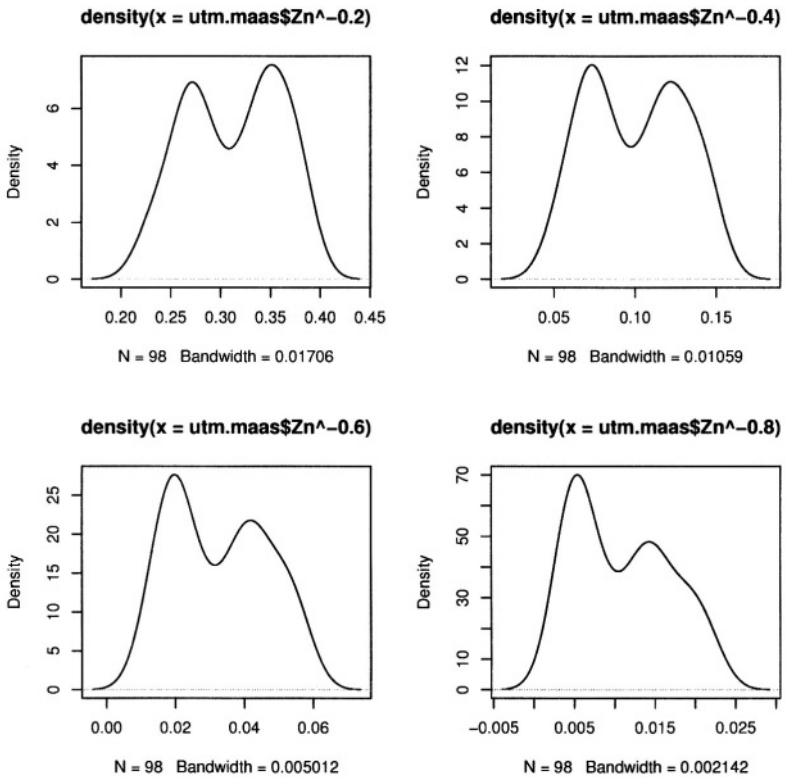


Figure 13.11. R/GRASS: Maas river bank soil data: Distribution of power-transformed zinc contamination data (various exponents)

Data transformations and trend surfaces. For geospatial analysis, data sets with skewed distribution are usually normalized (for a discussion see Bailey and Gatrell, 1995:172). As we have seen above, the zinc concentrations show a skewed distribution. We apply a logarithmic transformation, which is often used for normalization of geospatial data. A density plot of the log-transformed data shows a bimodal distribution (compare Figure 13.9). Now we want to try different approaches to normalize the data set, based on power transformation:

```
par(mfrow=c(2,2))
plot(density(utm.maas$Zn^-0.2))
plot(density(utm.maas$Zn^-0.4))
plot(density(utm.maas$Zn^-0.6))
plot(density(utm.maas$Zn^-0.8))
par(mfrow=c(1,1))
```

The resulting distribution curves are shown in Figure 13.11.

Before performing variogram modeling and kriging interpolation it is important to remove possible trends from data sets. As the zinc data probably contain a global trend, we apply a quadratic trend analysis (using Least Squares) to verify the situation:

```
library(spatial)
zinc <- data.frame(utm.maas$east, utm.maas$north, utm.maas$Zn)
names(zinc) <-  c("x", "y", "z")
zinc.ls2 <- surf.ls(2, zinc)
```

The function `surf.ls()` generates a trend model which is stored in object `zinc.ls2`. Next step is to evaluate a trend surface from this trend model by `trmat.G()` function. We can plot the resulting quadratic trend surface map `zinc.trend2`:

```
zinc.trend2 <- trmat.G(G, zinc.ls2)
plot(G, zinc.trend2)
points(zinc)
text(zinc$x, zinc$y, zinc$z, pos=1, cex=0.75)
```

Alternately we can also generate a cubic trend surface in a similar way:

```
zinc.ls3 <- surf.ls(3, zinc)
zinc.trend3 <- trmat.G(G, zinc.ls3)
plot(G, zinc.trend3)
points(zinc)
text(zinc$x, zinc$y, zinc$z, pos=1, cex=0.75)
```

Finally we write the cubic trend surface map to GRASS as raster map:

```
rast.put(G, lname="soilsph.cts", zinc.trend3)
q()
```

We can display this raster map with `d.rast`.

13.2.3 Using R in batch mode

`R` supports batch mode processing for a fully scripted usage. Within GRASS (maybe also scripted) geospatial data analysis can be automated. The desired analysis methods have to be stored in a text file (e.g. `R.trendph.batch`):

```
library(GRASS)
G <- gmeta()

#load map:
soilsph <- rast.get(G, "soils.ph", c(F))
names(soilsph) <- c("ph")
soils.ph.frame <- data.frame(east(G), north(G), soilsph$ph)
soilsph$ph[soilsph$ph == 0] <- NA
```

```
names(soils.ph.frame) <- c("x", "y", "z")

#calculate cubic trend surface:
library(spatial)
ph.ctrend <- surf.ls(3, na.omit(soils.ph.frame))
ph.ctrend.surf <- trmat.G(G, ph.ctrend)

#write plot to PDF:
pdf("trendSurf.pdf")
plot(G, ph.ctrend.surf, col=terrain.colors(20))
contour.G(G, ph.ctrend.surf, add=T)
title("Cubic trend surface of pH values in Spearfish region")
dev.off()

#cleanup workspace:
rm(list = ls(all = TRUE))
```

The last command is needed to avoid that the loaded data are stored in the R workspace file `.RData`. Alternately the flag `--no-save` can be used when running the script. The other commands used here will be known from previous sections. This script is run within GRASS (Spearfish LOCATION) through R batch mode:

```
grass53 /usr/local/share/grassdata/spearfish/user1
g.region -dpa res=100
R BATCH R.trendph.batch
cat R.trendph.batch.Rout
```

The function (and eventual error) messages are echoed in the file `R.trendph.batch.Rout` for batch process verification. In the example above, a plot of the trend surface in PDF format is included. You should find this file in the current directory if no error occurs.

The next example shows a batch job which calculates the empirical cumulative distribution function (ECDF) of a given map. Here we make use of environment variables so that we can write the script as a general script for any filename. Store the script as `R.ecdf.batch`:

```
#usage:
# export R_INMAP=rastermap
# R BATCH R.ecdf.batch
# -> ecdfplot.pdf

library(GRASS)
G <- gmeta()
library(stepfun)

#read input map from environment variable $R_INMAP:
map <- rast.get(G, Sys.getenv("R_INMAP"))
```

```
mapname <- Sys.getenv("R_INMAP")
names(map)<- c("z")

#generate data frame:
map.frame <- data.frame(east(G), north(G), map$z)
names(map.frame) <- c("x", "y", "z")

#write graph to PDF:
pdf("ecdfplot.pdf")

tstr <- c("ECDF (hypsometric integral):", mapname)
#plot ECDF:
mapecdf <- ecdf(map.frame$z)
plot(mapecdf,verticals=T,do.points=F,xlab="elevation",main=tstr)
for (i in seq(0,360,10)) lines(c(i,i),c(0,1),lty=2,col="red2")
dev.off()

#cleanup workspace:
rm(list = ls(all = TRUE))
```

To run it you have to define the GRASS raster map name to be analyzed in the environment variable `$R_INMAP`. We define it before starting the batch job (here for `bash` shell):

```
grass53 /usr/local/share/grassdata/spearfish/user1
g.region -dpa res=30
export R_INMAP=elevation.dem
R BATCH R.ecdf.batch
cat R.ecdf.batch.Rout
```

The resulting PDF file `ecdfplot.pdf` contains the graph showing the hypsometric integral of the input map (here: `elevation.dem`). When using above approach with environment variables, complex (pseudo) GRASS scripts can be written to extend functionality of GRASS by `R`.

13.3. GPS DATA HANDLING

The FreeGIS Web portal lists a set of programs freely available to handle GPS data. A basic problem to be addressed is the data transfer from GPS device to GIS including eventual datum transformations. These issues heavily depend on the GPS device. We only refer to a few software packages:

- GPS Manager (GPSMan[11]) is a graphical manager of GPS data that makes possible the preparation, inspection and edition of GPS data in a friendly environment. GPSMan supports communication and real-time logging with both Garmin and Lowrance receivers and accepts real-time logging information in NMEA 0183 from any GPS receiver;

- gpspoint[12] is a program to get position, down- and up-load waypoints, routes and tracks form your GPS to your computer. Several Garmin devices are supported by gpspoint;
- GPStrans[13] is a program which allows for track, route, and waypoint data to be transferred to and from various Garmin GPS;
- GPSBabel[14] reads and writes GPS waypoints in a variety of formats. Backends include GPX, Magellan and Garmin serial protocols, Geocaching.com, GPSMan, Garmin Mapsource, Magellan Mapsend, and many others. It runs on various operating systems.

GRASS itself provides two scripts: `v.in.garmin.sh` as well as `s.in.garmin.sh` (both require `gpstrans`). However, no checks are performed in these scripts for datum, projection and format of data. You must check by yourself that your receiver, `gpstrans` and GRASS use the same map datum and projection.

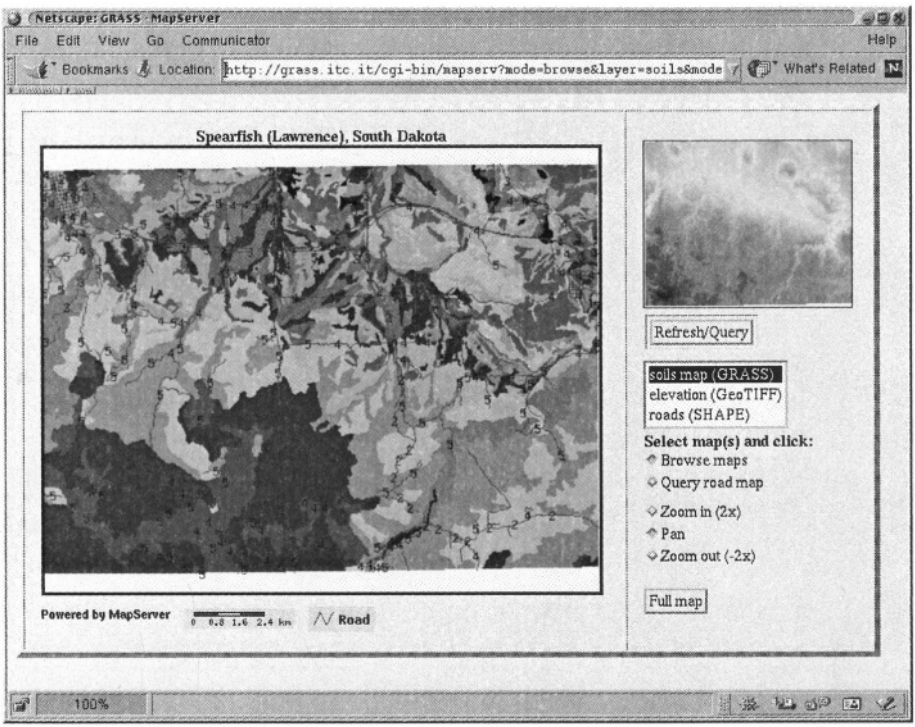


Figure 13.12. Screenshot of GRASS / UMN/MapServer demonstrational Web site as implemented at ITC-irst

13.4. WEBGIS APPLICATIONS WITH UMN/MapServer

An excellent, fast and flexible Open Source mapping Web software is UMN/MapServer. On a basic level, the program is run through CGI (Common Gateway Interface). Then it only requires a definition file and a HTML template (and GIS data of course) to respond to a variety of spatial requests like making maps, scale-bars, and point, area and feature queries. The installation is quite convenient as the configuration for the Web mapping interface can be done without any programming. For more complex applications, UMN/MapServer can be enhanced using Java, JavaScript, PHP or other Internet technologies. The MapScript extension which is based on PERL provides access to the underlying UMN/MapServer C API. Developers may add mapping functions to their PERL scripts. With MapScript also SHAPE files can be read or written. The UMN/MapServer software is freely available from the UMN/MapServer Web site.[15] It can be extended with UMN/MapServer Applets[16] to add menu icons into the map or other extended features which require JAVA.

In addition, with GDAL and OGR libraries (shipped with GDAL), UMN/MapServer reads common GIS raster and vector formats. When GDAL was compiled with "libgrass" support (GDAL/libgrass page[17]), UMN/MapServer directly reads raster data from a GRASS LOCATION through GDAL. With future releases of "libgrass" also vector and site data may become supported. Figure 13.12 provides a screenshot of the simple demonstrational GRASS / UMN/MapServer which is implemented at GRASS Web site.[18]

A more complex implementation using several Free Software tools is shown in Figure 13.13. Requirements for this implementation are: Web server such as Apache Server (with PHP), UMN/MapServer, GDAL/OGR, PROJ4, libgrass, GRASS and PostgreSQL/PostGIS. At the time of writing this book there have been limitations to automatically color-resample those GRASS raster data which contain more than 256 colors. If you intend to directly read from GRASS LOCATIONs, consider to rescale the map range to 8bit which is mostly sufficient for Web presentations (be sure not to use these maps in GIS computations!). The GRASS module for rescaling is `r.rescale`.

To read raster data directly from a GRASS LOCATION, you need, as mentioned above, the "libgrass" which is available on the GRASS Web site and CVS. This library reads the `.grassrc5` file for definition of LOCATION, MAPSET and GISDBASE (compare also Section 11.3). The file has to be stored in the `$DOCUMENT_ROOT` directory (might be the `$HOME` of the user-ID under which the local Web server is running). Due to GRASS permissions handling, the GRASS LOCATION data for UMN/MapServer have to be stored with the same user-ID as the Web server is running.

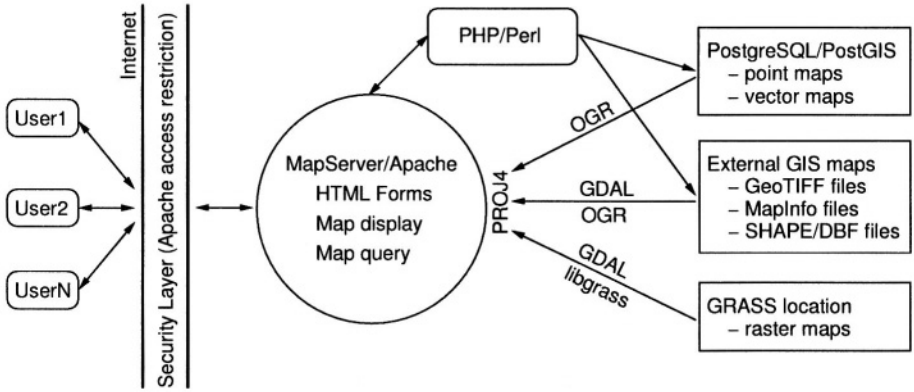


Figure 13.13. Sample UMN/MapServer implementation model

Two more files are required:

- UMN/MapServer definition file: to be stored in a `map-script` directory in parallel to the `htdocs/` directory (see sample in the Appendix C.1);
- a HTML template file which goes into the HTML space (into `htdocs/` directory, see sample in the Appendix C.2).

Additionally you may want to add further GIS data in different formats (remember to verify the data copyrights when publishing data online).

The minimum requirements for a UMN/MapServer implementation are a running Web server such as Apache, and appropriate access to the files. Once running, you may want to enrich the server with JAVA or PHP, numerous public mapservers are accessible on the Internet to get inspired.

NOTES

1 FreeGIS Project Web site, `http://www.freegis.org`
2 Maas river bank soil pollution data descriptions: `gstat` package documentation; Burrough and McDonnell, 1998:309-311 (subset)
3 Maas river bank soil data GRASS LOCATION, `http://grass.itc.it/statsgrass/maas_grass_location.tar.gz`
4 gstat software, `http://www.gstat.org`
5 gnuplot software, `http://gnuplot.sourceforge.net`
6 gstat examples, `http://www.gstat.org/examples.html`

7 R software, http://www.r-project.org
8 R Newsletter, http://cran.r-project.org/doc/Rnews/
9 R/GRASS interface,
http://cran.r-project.org/src/contrib/
10 geoR software,
http://www.est.ufpr.br/geoR/
11 GPS Manager, http://www.ncc.up.pt/gpsman/
12 gpspoint,
http://gpspoint.dnsalias.net/
13 GPStrans, http://sourceforge.net/projects/gpstrans
14 GPSBabel, http://sourceforge.net/projects/gpsbabel
15 UMN/MapServer project, http://mapserver.gis.umn.edu
16 UMN/MapServer Applets,
http://www2.dmsolutions.ca/mapserver/
17 GDAL/libgrass page
http://www.remotesensing.org/gdal/frmt_grass.html
18 Simple demonstrational GRASS / UMN/MapServer (Spearfish data),
http://grass.itc.it/start.html

References

Abramowitz, M., and I.A. Stegun, 1964. *Handbook of Mathematical Functions.* New York: Dover, 297-300, 228-231.

Albertz, J., 1991. *Grundlagen der Interpretation von Luft- und Satellitenbildern: Eine Einführung in die Fernerkundung.* Darmstadt: Wissenschaftliche Buchgesellschaft.

Albrecht, J., 1992. *GTZ-handbook GRASS.* Univ. of Vechta.
`http://grass.itc.it/gdp/`

Alexandrov, A.D., A.N. Kolmogorov, M.A. Lavrent'ev (eds), 1989. *Mathematics. Its content, methods, and meanings.* Vol. 2, 6th print. Cambridge (MA): MIT Press.

Arge, L., J.S. Chase, P. Halpin, L. Toma, J.S. Vitter, D. Urban, and R. Wickremesinghe, 2003. *Efficient Flow Computation on Massive Grid Terrain Datasets,* GeoInformatica, 7(4), 283-313.

Bailey, T.C., and A.C. Gatrell, 1995. *Interactive spatial data analysis.* Essex: Pearson.

Baker, W.L., and Y. Cai, 1992. The r.le programs for multiscale analysis of landscape structure using the GRASS geographical information system. Landscape Ecology, 7(4), 291-302.

Baker, W.L., 2001. The r.le programs. A set of GRASS programs for the quantitative analysis of landscape structure Version 5.0. Tech. report, Department of Geography and Recreation, University of Wyoming, 11/2001.

Bartelme, N., 1995. *Geoinformatik. Modelle, Strukturen, Funktionen.* Berlin: Springer.

Becker, R.A., J.M. Chambers, and A.R. Wilks, 1988. *The New S Language.* London: Chapman & Hall.

Bierhals, E., 1988. *CIR-Luftbilder für die flächendeckende Biotopkartierung. Informationsdienst Naturschutz Niedersachsen 5/88.* Hannover, 77-104.

Bill, R., 1996. *Grundlagen der Geo-Informations-Systeme. Analysen, Anwendungen und neue Entwicklungen.* Vol. 2, Heidelberg: Wichmann.

Bivand, R.S., 2000. Using the R statistical data analysis language on GRASS 5.0 GIS database files. Computers & Geosciences, 26, 1043-1052.

Bivand, R.S., and A. Gebhardt, 2000. Implementing functions for spatial statistical analysis using the R language. Journal of Geographical Systems, 3(2), 307-317.

Bivand, R.S., and M. Neteler, 2000. *Open Source geocomputation: using the R data analysis language integrated with GRASS GIS and PostgreSQL data base systems.* Proc. 5th conference on GeoComputation, 23-25 August 2000, University of Greenwich, U.K.
`http://reclus.nhh.no/gc00/gc009.htm`

Blazek, R., M. Neteler and R. Micarelli, 2002. *The new GRASS 5.1 vector architecture.* Proc. Open Source GIS – GRASS users conference, 11-13 September 2002, Trento, Italy.

Brandon, R.J., T. Kludt, and M. Neteler, 1999. Archaeology and GIS – The Linux Way. Using GRASS and Linux to analyze archaeological data. Feature article, Linux Journal (7), 50-54.

Brown, W.M., M. Astley, T. Baker, and H. Mitasova, 1995. GRASS as an integrated GIS and visualization environment for spatio-temporal modeling. Proc. Auto-carto XII, ACSM/ASPRS, Charlotte, NC, 89-99.
http://www2.gis.uiuc.edu:2280/modviz/brown/papers/AC12.html

Bugayevskiy, L.M, and J.P. Snyder 2000. *Map Projections. A reference manual.* London, Philadelphia: Taylor & Francis.

Burns, L.E., M.B. Werdon, and R.J. Newberry, 2000. *Evaluation of Satellite and Airborne Synthetic Aperture Radar Data for an Area Northeast of Fairbanks, Alaska.* Electronic Document: http://www.astf.aeromap.com/dggs_fr.html

Burrough, P.A., 1986. *Principles of GIS for land resources assessment.* Oxford: Clarendon Press.

Burrough, P.A., and R.A. McDonnell, 1998. *Principles of Geographical Information Systems.* New York: Oxford University Press.

Chambers, J.M., and T.J. Hastie, 1992. *Statistical Models in S.* London: Chapman & Hall.

Chavez, P.C., S.C. Guptill, and J.A. Bowell, 1984. *Image processing techniques for Thematic Mapper data.* Proc., Am. Soc. Photogr. 2, 728-743.

Chavez Jr., P.S., 1996. Image-based atmospheric corrections – revisited and improved. Photogr. Eng. & Rem. Sens., 62(9), 1025-1036.

Clarke, K.C., 2002. *Getting started with Geographic Information Systems.* 4th ed. New Jersey: Prentice Hall.

Clarke, K.C., B. Parks, and M. Crane, (eds) 2002. *Geographic Information Systems and Environmental Modeling.* Upper Saddle River, New York: Prentice Hall.

Cressie, N.A.C., 1993. *Statistics for spatial data.* New York: Wiley.

Dana, P.H., 2000. *Map projections, coordinate systems. The Geographer's Craft Project.* Department of Geography, The University of Colorado at Boulder.
http://www.colorado.edu/geography/gcraft/notes/mapproj/mapproj_f.html
http://www.colorado.edu/geography/gcraft/notes/coordsys/coordsys_f.html

Desmet, P.J.J., and G. Govers, 1996. A GIS procedure for automatically calculating the USLE LS factor on topographically complex landscape units. J. Soil and Water Conservation, 51(5), 427-433.

Dikau, R., 1989. The application of a digital relief model to landform analysis in geomorphology. In Raper, J. (ed), *Three dimensional applications in Geographic Information Systems.* London: Taylor & Francis, 51-77.

Evenden, G., 1995. *Cartographic Projection Procedures for the Unix Environment – A User's Manual.* U.S. Geological Survey Open File Report 90-284.
http://www.remotesensing.org/proj/

Fortune, S.J., 1987. A Sweepline Algorithm for Voronoi Diagrams. Algorithmica 2, 153-174.

Furlanello, C., M. Neteler, S. Merler, S. Menegon, S. Fontanari, A. Donini, A. Rizzoli, and C. Chemini, 2003. GIS and the randomForest Predictor: integration in R for tick-borne disease risk assessment. Proceedings of "Distributed Statistical Computing 2003", Eds: Hornik, K., Leisch, F., Vienna, Austria, 20-22 March 2003.

Goodchild, M.F, L.T. Steyaert, and B.O. Parks (eds), 1993. *Geographic Information Systems and Environmental Modeling.* New York: Oxford University Press.

Goodchild, M.F., L.T. Steyaert, and B.O. Parks (eds), 1996. *GIS and Environmental Modeling: Progress and Research Issues.* Ft. Colins: GIS World, Inc.

Goodchild, M.F., L.T. Steyaert, and B.O. Parks (eds), 1997. *GIS and Environmental Modeling.* Proceedings of the 3rd conference on GIS and Environmental Modeling. Santa Fe: NCGIA, CDROM.

GSFC/NASA, 2001. Landsat-7 Science Data User's Handbook. Electronic Document: `http://ltpwww.gsfc.nasa.gov/IAS/handbook/handbook_htmls/chapter11/chapter11.html`

Haan, C.T., B.J. Barfield, and J.C. Hayes, 1994. *Design Hydrology and Sedimentology for Small Catchments.* New York: Academic Press.

Hake, G., and D. Grünreich, 1994. *Kartographie.* 7th ed. Berlin: de Gruyter.

Hibbard, W. L., B.E. Paul, D.A. Santek, C.R. Dyer, A.L. Battaiola, and M.-F. Voidrot-Martinez, 1994. Interactive Visualization of Earth and Space Science Computations. Computer 27(7), July 1994, 65-72.

Hildebrandt, G., 1996. *Fernerkundung und Luftbildmessung: für Forstwirtschaft, Vegetationskartierung und Landschaftsökologie.* Heidelberg: Wichmann.

Hofierka, J., 1997a. Modeling natural phenomena in a GIS environment. Ph.D. dissertation (in Slovak), Comenius University, Bratislava, p. 83.

Hofierka, J., 1997b. *Direct solar radiation modelling within an open GIS environment.* In: Hodgson, S., M. Rumor, and J.J. Harts (eds). *Geographical Information '97: Third Joint European Conference & Exhibition on Geographical Information.* Proc., Vienna, Austria, April 1997, 1,575-584.

Hofierka J., J. Parajka, H. Mitasova, and L. Mitas, 2002. Multivariate Interpolation of Precipitation Using Regularized Spline with Tension. Transactions in GIS, 6(2), 135-150.

Hofierka, J. and M. Súri, 2002. The solar radiation model for Open source GIS: implementation and applications. Proceedings of the "Open Source Free Software GIS – GRASS users conference 2002", Eds: Ciolli, M. and P. Zatelli, Trento, Italy, 11-13 September 2002. CD-ROM. p. 19.

Horn, B.K.P., 1981. Hill Shading and the Reflectance Map. Proceedings of the IEEE, 69(1), 14-47.

Hutchinson, M.F., R.J. Bischof, 1983. A new method for estimating the spatial distribution of mean seasonal and annual rainfall applied to the Hunter Valley, New South Wales. Australian Meteorological Magazine 31, 179-184.

Hutchinson, M.F. , 1991. The Application of thin plate smoothing splines to continent-wide data assimilation. In: *Data Assimilation Systems,* Jasper, J.D. (ed), BMRC Research Report N.27, Bureau of Meteorology, Melbourne, 104-113.

Ihaka, R., and R. Gentleman, 1996. R: A Language for Data Analysis and Graphics. J. of Comp. and Graph. Stat., (5)3, 299-314.

Jacoby, W.G., 1997. *Statistical graphics for univariate and bivariate data.* Sage university papers: Series 7, Quantitative applications in the social sciences (117). Thousand Oaks, California: Sage.

Journel A.G., 1996. Modelling uncertainty and spatial dependence: Stochastic imaging. Int. J. of Geogr. Inf. Sys. 10(5), 517-22.

Kasten, F., 1996. The Linke turbidity factor based on improved values of the integral Rayleigh optical thickness. Solar Energy, 56(3), 239-244.

Kramer, H.J., 1996. *Observation of the earth and its environment: survey of missions and sensors.* 3rd ed. Berlin: Springer.

Krcho, J., 1973. *Morphometric analysis of relief on the basis of geometric aspect of field theory.* Acta Geographica Universitae Comenianae, Geographica Physica 1, Bratislava, SPN.

Krcho, J., 1991. Georelief as a subsystem of landscape and the influence of morphometric parameters of georelief on spatial differentiation of landscape-ecological processes. Ecology /CSFR/ (10), 115-157.

Longley, P.A., M. Goodchild, D.J. Maguire, and D.W. Rhind, 2002, *Geographic Information Systems and Science.* London: Wiley.

McCauley, J.D., and B.A. Engel, 1995. Comparison of Scene Segmentations: SMAP, ECHO and Maximum Likelihood. IEEE Trans. on Geosc. & Rem. Sens., 33(6): 1313-1316.

Maling, D.H., 1992. *Coordinate Systems and Map Projections.* 2nd ed. Elmsford, New York: Pergamon Press.

Mandelbrot, B.B., 1983. *The fractal geometry of nature.* New York: Freeman.

Mather, P.M., 1999. *Computer processing of remotely-sensed images.* Chichester: Wiley.

Mitas, L., W.M. Brown, and H. Mitasova, 1997. Role of dynamic cartography in simulations of landscape processes based on multi-variate fields. Comp. & Geosc., 23, 437-446. `http://skagit.meas.ncsu.edu/~helena/gmslab/lcgfin/cg-mitas.html`

Mitas, L., and H. Mitasova, 1999. Spatial Interpolation. In: P. Longley, M.F. Goodchild, D.J. Maguire, and D.W. Rhind (eds), *Geographical Information Systems: Principles, Techniques, Management and Applications.* New York: Wiley, 481-492.

Mitasova, H., and L. Mitas, 2002. Modeling Physical Systems. In: K.C. Clarke, B. Parks, and M. Crane (eds), *Geographic Information Systems and Environmental Modeling.* Prentice Hall, 189-210.

Mitasova, H., and L. Mitas, 2001. Multiscale soil erosion simulations for land use management, In: R.S. Harmon, and W.W. Doe (eds), *Landscape erosion and landscape evolution modeling.* New York: Kluwer Academic/Plenum Publishers, 321-347.

Mitasova, H., J. Hofierka, M. Zlocha, L.R. Iverson, 1996. Modeling topographic potential for erosion and deposition using GIS. Int. J. of Geogr. Inf. Sci., 10(5), 629-641.

Mitasova H., L. Mitas, W.M. Brown, D.P. Gerdes, I. Kosinovsky, and T. Baker, 1995. Modeling spatially and temporally distributed phenomena: New methods and tools for GRASS GIS. Int. J. of GIS, 9(4), Special issue on Integrating GIS and Environmental modeling, 433-446.

Mitasova H., and L. Mitas, 1993. Interpolation by Regularized Spline with Tension: I. Theory and Implementation. Math. Geol. 25, 641-655.

Mitasova H., and J. Hofierka, 1993. Interpolation by Regularized Spline with Tension: II. Application to Terrain Modeling and Surface Geometry Analysis. Math. Geol. 25, 657-667.

Monmonier, M., 1996. *How to Lie with Maps.,* 2nd ed. Chicago: University of Chicago Press.

Moore, I.D., A.K. Turner, J.P. Wilson, S.K. Jensen, and L.E. Band, 1992. GIS and land surface-subsurface process modeling. In: Goodchild, M.F., B. Parks, and L.T. Steyaert (eds), *Geographic Information Systems and Environmental Modeling.* Oxford University Press, New York.

Moore I.D., and G.J. Burch, 1986. Modeling erosion and deposition: Topographic effects. Transactions ASAE, 29, 1624-1640.

Moore, I.D., R.B. Grayson, and A. R. Ladson, 1991. Digital terrain modelling: a review of hydrological, geomorphological and biological applications. Hydrol. Processes, 5, 3-30.

Moore, I.D., and Wilson, J.P., 1992. Length-slope factors for the Revised Universal Soil Loss Equation: Simplified method of estimation. Journal of Soil and Water Conservation, 47, 423-428.

Moran, M.S., R.D. Jackson, P.N. Slater and P.M. Teillet, 1992. Evaluation of simplified procedures for retrieval of land surface reflectance factors from satellite sensor output. Rem. Sens. Env. 41, 169-184.

NCGIA, 2000. The NCGIA Core Curriculum in GIScience. Electronic Document: `http://www.ncgia.ucsb.edu/giscc/`

Neteler, M., 1999. *Spectral Mixture Analysis von Satellitendaten zur Bestimmung von Bodenbedeckungsgraden im Hinblick auf die Erosionsmodellierung.* M.Sc. Thesis, Univ. of Hannover.

Neteler, M., 2000. *GRASS-Handbuch. Der praktische Leitfaden zum Geographischen Informationssystem GRASS.* Geosynthesis 11, Univ. of Hannover.
`http://grass.itc.it/gdp/handbuch/`

Neteler, M., 2001a. *Towards a stable open source GIS: Status and future directions in GRASS development.* In: Brovelli, M. (ed), 2001. *The Geomatics Workbook N. 2.* Polytec. di Milano. Electronic Document:
`http://geomatica.ing.unico.it/workbooks2/index.html`

Neteler, M., 2001b. *Volume modeling of soils using GRASS GIS 3D tools.* In: Brovelli, M. (ed), 2001. *The Geomatics Workbook N. 2.* Polyt. di Milano. Electronic Document:
`http://geomatica.ing.unico.it/workbooks2/index.html`

Neteler, M. (ed), 2002. *GRASS 5.0 Programmer's Manual. Geographic Resources Analysis Support System.* ITC-irst, Trento. Electronic Document:
`http://grass.itc.it/grassdevel.html`

Oliver, C., and S. Quegan, 1998. *Understanding Synthetic Aperture Radar Image.* London: Artech House.

O'Rourke, J., 1998. *Computational Geometry in C.* 2nd ed., Cambridge: Cambridge University Press.

Page, J., M. Albuisson, L. Wald, 2001. The European solar radiation atlas: a valuable digital tool. Solar energy, 71, 81-83.

Pebesma, E.J., and C.G. Wesseling, 1998. Gstat: a program for geostatistical modelling, prediction and simulation. Comp. & Geosc. 24(1), 17-31.
`http://www.gstat.org`

Pebesma, E.J. 2001. *Gstat user's manual. gstat 2.3.3.* Electronic document:
`http://www.gstat.org`

Peek, J., G. Todino, and J. Strang, 2001. *Learning the Unix Operating System. A Concise Guide for the New User.* 5th ed. Cambridge: O'Reilly & Associates.

Pohl, C., and J.L. van Genderen, 1998. Multisensor image fusion in remote sensing: concepts, methods and application. Int. J. of Rem. Sens., 19, 823-854.

Powell M.J.D., 1992. Tabulation of Thin Plate Splines on a very fine two-dimensional grid. In Braess D., and L.L. Schumaker (eds), *Numerical Methods of Approximation Theory* 9: 221-44.

Rase, W.D., 1998. *Visualisierung von Planungsinformationen: Modellierung und Darstellung immaterieller Oberflächen.* Bundesamt für Bauwesen und Raumordnung, Forschungen H. 89, Bonn.

Raymond, E., 1997. *The cathedral and the bazaar.* Electronic document:
`http://www.ccil.org/ esr/writings/cathedral-paper.html`

Raymond, E., 1999. *The cathedral and the bazaar. Musings on Linux and Open Source by an accidental revolutionary.* Cambridge: O'Reilly & Associates.

Redslob, M., 1998. Radarfernerkundung in niedersächsischen Hochmooren. Diss. Inst. f. Landschaftspfl. u. Natursch., Univ. of Hannover.

Rektorys, K., 1969. *Survey of Applicable Mathematics.* Cambridge, MA: MIT Press and London: Iliffe Books Ltd., 365.

Ribeiro, J.R., and P.J. Diggle, 1999. *geoS: A geostatistical library for S-PLUS.* Technical report ST-99-09, Dept of Math. and Stat., Lancaster University.
`http://www.maths.lancs.ac.uk/~ribeiro/geoR.html`

Ribeiro, J.R., and P.J. Diggle, 2001. geoR: A package for geostatistical analysis. R-NEWS (1)2, 15-18. Electronic document:
`http://cran.r-project.org/doc/Rnews`

Rigollier, Ch., O. Bauer, L. Wald, 2000. On the clear sky model of the ESRA - European Solar radiation Atlas - with respect to the Heliosat method. Solar energy, 68, 33-48.

Ripley, B.D., 1996. *Pattern recognition and neural networks.* Cambridge: Cambridge University Press.

Richards, J.A., and J. Xiuping 1999. *Remote sensing digital image analysis: An introduction.* 3rd ed. Heidelberg: Springer.

Robbins, A., and D. Gilly, 1999. *Unix in a Nutshell: A Desktop Quick Reference for SVR4 and Solaris 7.* 3rd ed. Cambridge: O'Reilly & Associates.

Robinson, A.H., J.L. Morrison, P.C. Muehricke, A.J. Kimerling, and S.C. Guptill, 1995. *Elements of Cartography.* Reprint of 6th ed. New York: Wiley.

Sandmeier, S., 1995. *A physically-based radiometric correction model. Correction of atmospheric and illumination effects in optical satellite data of rugged terrain.* Diss. Geogr. Inst., Univ. Zürich.

Scharmer, K., and J. Greif (eds), 2000. The European solar radiation atlas. Vol. 2: Database and exploitation software. Paris: Les Presses de l'École des Mines.

Schowengerdt, R., 1997. *Remote sensing: Models and methods for image processing.* 2nd ed. San Diego: Academic Press.

Shapiro, M., and J. Westervelt 1992. *r.mapcalc. An algebra for GIS and image processing.* US-Army CERL, Champaign, Illinois, 422-425.
`http://grass.itc.it/gdp/`

Siever, E. (ed), J.P. Hekman, S. Figgins, and S. Spainhour, 2000. *LINUX in A Nutshell: A Desktop Quick Reference.* 3rd ed. Cambridge: O'Reilly & Associates.

Singh, S.M., 1988. *Brightness temperature algorithms for Landsat Thematic Mapper data.* Rem. Sens. Env., 24, 509-512.

Snyder, J.P. 1987. *Map Projections, A working manual.* U.S. Geological Survey Professional Paper 1395. Department of the Interior. Washington, D.C.

Súri, M., and J. Hofierka, 2004. A new GIS-based solar radiation model and its application to photovoltaic assessments. Transactions in GIS, 8(2), 175-190.

Talmi, A., and G. Gilat, 1977. Method for Smooth Approximation of Data. *J. of Computational Physics,* 23, 93-123.

Teillet, P.M., B. Guindon, and D.G. Goodenough, 1982. *On the slope-aspect correction of multispectral scanner data* Canad. J. Rem. Sens., 8(2), 36-44.

Ulaby, F.T., R.K. Moore, and A.K. Fung, 1982. *Microwave remote sensing: Active and passive.* Vols 1, 2, 3. Reading (Mass.): Addison-Wesley.

U.S. Army CERL, 1993. *GRASS4.1 Reference Manual.* U.S. Army Corps of Engineers, Construction Engineering Research Laboratories, Champaign, Illinois, 1-425.

Venables, W.N., and B.D. Ripley, 2000. *S programming.* New York: Springer.

Venables, W.N., and B.D. Ripley, 2002. *Modern applied statistics with S.* 4th ed. New York: Springer. Supplements:
`http://www.stats.ox.ac.uk/pub/MASS4/`

Vermote, E., D. Tanré, J.L. Deuzé, M. Herman, and J.J. Morcrette, 1997. Second Simulation of the Satellite Signal in the Solar Spectrum, 6S: An Overview. IEEE Trans. Geosc. Rem. Sens. 35(3), 675-686.

Wadsworth, R., and J. Treweek, 1999. *Geographical Information systems for Ecology: An Introduction.* Essex: Longman.

Wahba, G., 1990. *Spline models for observational data.* CNMS-NSF Regional Conference series in applied mathematics, 59, SIAM, Philadelphia, Pennsylvania.

Watson, D. F. 1992. *Contouring: a guide to the analysis and display of spatial data.* Oxford: Pergamon.

Watson, K., 1993. Processing remote sensing images using the 2-D FFT-noise reduction and other applications. Geophysics, 58(6), 835-852.

Webster, R., and M.A. Oliver, 2001. *Geostatistics for environmental scientists.* Chichester: Wiley.

Wessel, P., and W.H.F. Smith, 1996. A Global Self-consistent, Hierarchical, High-resolution Shoreline Database. J. of Geophys. Res., 101, 8741-8743.

Wheeler, D., 2003. *Why Open Source: Look at the numbers!* Electronic Document: `http://www.dwheeler.com/oss_fs_why.html`

Wilson, J.P., and J.C. Gallant, 2000. *Terrain Analysis: Principles and Applications.* New York: Wiley.

Wood, J.D., and P.J. Fisher, 1993. Assessing Interpolation Accuracy in Elevation Models. IEEE Comp. Graph. and Appl., 13(2), 48-56.

Wood, J., 1996. *The Geomorphological characterisation of Digital Elevation Models.* Diss., Dep. of Geogr., Univ. of Leicester, U.K. `http://www.geog.le.ac.uk/jwo/research/dem_char/thesis/`

Yeung, A.K., 1998. *Unit 051 - Information Organization and Data Structure,* NCGIA Core Curriculum in GIScience, NCGIA. Electronic Document: `http://www.ncgia.ucsb.edu/giscc/units/u051/`

Zevenbergen, L.W., and C.R. Thorne, 1987. Quantitative analysis of land surface topography. Earth Surf. Proc. and Landf., 12, 47-56.

Zhou, J., D.L. Civico, and J.A. Silander, 1998. A wavelet transform method to merge LANDSAT-TM and SPOT panchromatic data. Int. J. of Rem. Sens., 19(4), 743-757.

This page intentionally left blank

Appendix A
Using UNIX text tools for GIS data preparation

The GNU text tools `cat`, `cut`, `join`, `head`, `more`, `paste`, `sed` and `tail` and the `awk` ("pattern scanning and processing language") provide a range of possibilities to modify ASCII texts and tables. Often attribute tables are delivered in ASCII formats such as CSV (Comma Separated Values format) or blank delimited text. Especially in scripts the tools introduced here are quite helpful to automate text formatting.

In a small sample session we show modifications of the Spearfish soils map legend which is available at GRASS Web site and already included in the Spearfish sample data set (see `/usr/local/share/grassdata/spearfish/soils_legend.txt`). This legend is an ASCII table, which contains further attributes for the soils map. We want to show how to modify this table to a reclass rules file applicable to `v.reclass` and `r.reclass`. First let's have a look into the file:

```
more soils_legend.txt
```

Within the `more` program continue to a next page with <SPACE>, `quit` with `q`, search for a phrase with /. Above file may look like this:

```
0:no data:
1:AaB:Alice fine sandy loam, 0 to 6
2:Ba:Barnum silt loam
3:Bb:Barnum silt loam, channeled
4:BcB:Boneek silt loam, 2 to 6
5:BcC:Boneek silt loam, 6 to 9
6:BeE:Butche stony loam, 6 to 50
[...]
```

The legend columns are separated by ":". In the first column the category numbers (attribute IDs) are stored. In the second column the first letter always capital is the initial letter of the soil name. The second letter is a capital if the mapping unit is broadly defined; otherwise, it is a small letter. The third letter, always a capital, A, B, C, D, E or F, indicates the slope. Symbols without slope letter are those of mapping units that do not have slope as part of the name. In the third column the full name of the soil is written along with the typical slope.

First we want to reduce the legend to attribute ID, soil name initials and text attribute without the slope information (this may go into another table or derived from the Spearfish

elevation.dem). Note that we proceed step-by-step although you can also compose the commands to a few (or even a single) lines. The cut tool cuts column-wise depending on the specified delimiter. We specify delimiter : and select the first column (with field parameter f), then pipe the result into a new file:

```
cut -d',' -f1 soils_legend.txt >  soils_legend2.txt
```

Checking the new file with more shows us:

```
0:no data:
1:AaB:Alice fine sandy loam
2:Ba:Barnum silt loam
3:Bb:Barnum silt loam
4:BcB:Boneek silt loam
5:BcC:Boneek silt loam
6:BeE:Butche stony loam
[...]
```

Starting from the new file we will select only the text label:

```
cut -d':' -f3 soils_legend2.txt > soils_legendlabels.txt
```

Checking the new file with more shows us:

```
Alice fine sandy loam
Barnum silt loam
Barnum silt loam
Boneek silt loam
Boneek silt loam
Butche stony loam
[...]
```

Note that the first line is empty since the no-data field doesn't contain a text label. Alternatively you can compose above steps to one command:

```
cut -d',' -f1 soils_legend.txt | cut -d':' -f3 >\
                              soils_legendlabels.txt
```

Now further hints: In case you want to cut a column at a specific position, you can use the -b flag:

```
cut -b1,2 soils_legend.txt

0:n
1:A
2:B
3:B
4:B
5:B
6:B
[...]
```

If you want to see only the first lines of a file, use head. The number of lines to be displayed has to be entered with a preceding minus character:

```
head -4 soils_legend.txt
```

The result looks as follows:

```
0:no data:
1:AaB:Alice fine sandy loam, 0 to 6
2:Ba:Barnum silt loam
3:Bb:Barnum silt loam, channeled
```

If you want to see only the last lines, use `tail`. It is used similar to `head`:

```
tail -3 soils_legend.txt
```

leading to the output:

```
53:WaA:Weber loam, 0 to 2
54:Wb:Winetti cobbly loam
55:water
```

Both may be combined to show a portion of the text file (here only lines 3 to 5):

```
head -5 soils_legend.txt | tail -3
```

Here the `head` command shows the first five lines, the `tail` command the last three of these five lines:

```
2:Ba:Barnum silt loam
3:Bb:Barnum silt loam, channeled
4:BcB:Boneek silt loam, 2 to 6
```

In order to sequentially concatenate two files, use `cat`:

```
cat file1 file2 > file1and2
```

If you need to paste two files column-wise, use `paste`. You can optionally change the column delimiter from the default tabulator to another character. The command `join` is allowing to work similar to a simple database management system – it joins together ASCII tables according to unique column entries.

A powerful string editor is `sed` which allows to exchange, add or cut off strings from text files by rule definitions.

With `awk` which we already used throughout the book, you can perform calculations or formatted printing. For details please refer to the related manual pages.

Appendix B
Selected equations used in GRASS modules

In this section we provide equations for selected GRASS modules, for those users who would like to have deeper understanding of the methods used in these modules and gain more confidence in advantages and limitations of the provided functionality. The equations are also helpful for those who would like improve or extend the modules.

While the number of modules for which we can provide the equations is currently limited we plan to extend this type of in depth description to other modules in future editions.

B.1. BASIC STATISTICS

Arithmetic Mean:

$$\mu = \frac{1}{n}(x_1 + x_2 + \ldots + x_n) = \frac{1}{n}\sum_{p=1}^{n} x_p \qquad \text{(B.1)}$$

Arithmetic Mean is not unitless.

Median:
The Median is the value below which 50% of the sample lie. To find the Median the data have to be ordered from smallest to highest. In case of an odd number of samples it is the middle value, in case of an even number of samples it is as half way between the two middle samples. Median is not unitless.

Variance:

$$\sigma^2 = \frac{1}{n-1}\sum_{p=1}^{n}(x_p - \mu)^2 \qquad \text{(B.2)}$$

Variance is not unitless.

Standard Deviation:

$$\sigma = \sqrt{\frac{1}{n-1}\sum_{p=1}^{n}(x_p - \mu)^2} \tag{B.3}$$

Standard Deviation is not unitless.

Coefficient of variation:

$$v = \frac{\sigma}{|\mu|} * 100 \tag{B.4}$$

Coefficient of Variation is unitless.

Skewness:

$$skewness = \frac{1}{n}\sum_{p=1}^{n}\left(\frac{x_p - \mu}{\sigma}\right)^3 \tag{B.5}$$

Skewness is zero for any symmetric distribution. A distribution with a long tail towards larger values has a positive skewness (left skewed, typical for remote sensing images, Schowengerdt, 1997: 118). Skewness is unitless and sensitive to outliers.

Kurtosis:

$$kurtosis = \left[\frac{1}{n}\sum_{p=1}^{n}\left(\frac{x_p - \mu}{\sigma}\right)^4\right] - 3 \tag{B.6}$$

Kurtosis is zero for a normal distribution. If a distribution has a positive kurtosis, than the peak is sharper than of a Gaussian distribution. Kurtosis is unitless and sensitive to outliers.

Covariance:

$$covariance = \frac{1}{n-1}\sum_{p=1}^{n}(x_{pm} - \mu_m)(x_{pn} - \mu_n) \tag{B.7}$$

B.2. INTERPOLATION

Inverse distance weighted interpolation (IDW). The method is based on an assumption that the value at an unsampled point can be approximated as a weighted average of values at points within a certain cut-off distance, or from a given number m of the closest points (typically 10 to 30). Weights are usually inversely proportional to a power of distance (Watson, 1992, Burrough, 1986) which, at an unsampled location $\mathbf{r} = (x,y)$, leads to an estimator

$$F(\mathbf{r}) = \sum_{i=1}^{m} w_i z(\mathbf{r}_i) = \frac{\sum_{i=1}^{m} z(\mathbf{r}_i)/|\mathbf{r} - \mathbf{r}_i|^p}{\sum_{j=1}^{m} 1/|\mathbf{r} - \mathbf{r}_j|^p} \tag{B.8}$$

where p is a parameter (typically $p = 2$, for more details on the influence of this parameter see Watson, 1992). GRASS modules use $p = 2$.

Regularized Spline with Tension. The function is a sum of a *trend function* and a *radial basis function* with an explicit form which depends on the choice of the measure of smoothness, for more details see Mitasova and Mitas, 1993, Mitasova et al., 1995:

$$z(\mathbf{r}) = T(\mathbf{r}) + \sum_{j=1}^{N} \lambda_j R(\mathbf{r}, \mathbf{r}^{[j]}) . \tag{B.9}$$

The trend function $T(\mathbf{r})$ is given by

$$T(\mathbf{r}) = \sum_{l=1}^{M} a_l f_l(\mathbf{r}) \tag{B.10}$$

where $\{f_l(\mathbf{r})\}$ is a set of linearly independent functions (monomials) which have zero smooth seminorm. $R(\mathbf{r}, \mathbf{r}^{[j]})$ is a radial basis function with an explicit form which depends on the choice of weights for partial derivatives in the smooth seminorm. See Mitasova and Mitas, 1993, Mitasova et al., 1995 for the RST smoothness seminorm, which includes derivatives of all orders with their weights decreasing with the increasing derivative order.

RST can be generalized to an arbitrary dimension and the corresponding d-variate formula for the radial basis function is given by

$$R_d(\mathbf{r}, \mathbf{r}_j) = R_d(|\mathbf{r} - \mathbf{r}_j|) = R_d(r) = \rho^{-\delta}\, \gamma(\delta, \rho) - \frac{1}{\delta} \tag{B.11}$$

where $r = |\mathbf{r} - \mathbf{r}_j|$, $\delta = (d-2)/2$, and $\rho = (\varphi r/2)^2$. Further, φ is a generalized tension parameter, and $\gamma(\delta, \rho)$ is the incomplete gamma function, not to be confused with semivariogram (Abramowitz and Stegun, 1964). For the special cases d = 2,3,4 (`s. surf.rst`, `s.vol.rst`, `s. volt. rst`, respectively), the equation B. 11 can be rewritten as:

$$R_2(r) = -[\mathrm{E}_1(\rho) + \ln\rho + C_E] \tag{B.12}$$

$$R_3(r) = \sqrt{\frac{\pi}{\rho}}\, \mathrm{erf}(\sqrt{\rho}) - 2 \tag{B.13}$$

$$R_4(r) = \frac{1 - e^{-\rho}}{\rho} - 1 \tag{B.14}$$

where $C_E = 0.577215...$ is the Euler constant, $\mathrm{E}_1(\rho)$ is the exponential integral function and $\mathrm{erf}(\sqrt{\rho})$ is the error function (Abramowitz and Stegun, 1964), while the trend function is a constant (M = 1):

$$T(\mathbf{x}) = a_1\,, \quad d = 2,3,4 \tag{B.15}$$

The coefficients $a_1, \{\lambda_j\}$ are obtained by solving the following system of linear equations

$$a_1 + \sum_{j=1}^{N} \lambda_j \left[R(\mathbf{r}^{[i]}, \mathbf{r}^{[j]}) + \delta_{ji} w_0 / w_j\right] = z^{[i]}\,, \quad i = 1, \ldots, N \tag{B.16}$$

$$\sum_{j=1}^{N} \lambda_j = 0. \tag{B.17}$$

B.3. TOPOGRAPHIC ANALYSIS

Topographic parameters slope, aspect and curvatures are computed using the principles of differential geometry using the work by Krcho, 1973, Krcho, 1991 and Mitasova and Hofierka, 1993. Before deriving mathematical expressions for these parameters, using the basic principles of differential geometry, the following simplifying notations are introduced:

$$f_x = \frac{\partial z}{\partial x}, \qquad f_y = \frac{\partial z}{\partial y}, \qquad f_{xx} = \frac{\partial^2 z}{\partial x^2}, \qquad f_{yy} = \frac{\partial^2 z}{\partial y^2}, \qquad f_{xy} = \frac{\partial^2 z}{\partial x \partial y} \tag{B.18}$$

and

$$p = f_x^2 + f_y^2, \quad q = p + 1 \quad . \tag{B.19}$$

The steepest slope angle γ and aspect angle α are computed from gradient $\nabla f = (f_x, f_y)$ (its direction is upslope) as follows

$$\gamma = \arctan\sqrt{p} \tag{B.20}$$

$$\alpha = \arctan\frac{f_y}{f_x} \qquad (\alpha = 0 \text{ in west direction}) \tag{B.21}$$

Sometimes we need to compute change of the surface in a direction given by an angle α. The directional derivative of the surface $z = g(x,y)$ can be computed as

$$E = \frac{\partial g}{\partial s} = \frac{\partial g}{\partial x}\cos\alpha + \frac{\partial g}{\partial y}\sin\alpha \tag{B.22}$$

where (x,y) are the georeferenced coordinates, and α is aspect (given direction).

Curvatures. In general, a surface has different curvatures in different directions. For applications in geosciences, the curvature in gradient direction (profile curvature) is important because it reflects the change in slope angle and thus controls the change of velocity of mass flowing down along the slope curve. The curvature in a direction perpendicular to the gradient (tangential curvature) reflects the change in aspect angle and influences the divergence/convergence of water flow. Both curvatures are measured in the normal plane. Equations for these curvatures can be derived using the general equation forcurvature κ of a plane section through a point on a surface (Rektorys, 1969, Mitasova and Mitas, 1993).

The equation for the profile curvature is

$$\kappa_s = \frac{f_{xx}f_x^2 + 2f_{xy}f_xf_y + f_{yy}f_y^2}{p\sqrt{q^3}} \quad . \tag{B.23}$$

The equation for tangential curvature κ_t at a given point is derived as the curvature of normal plane section in a direction perpendicular to gradient (direction of tangent to the contour line)

$$\kappa_t = \frac{f_{xx}f_y^2 - 2f_{xy}f_xf_y + f_{yy}f_x^2}{p\sqrt{q}} \quad . \tag{B.24}$$

The positive and negative values of profile and tangential curvature can be combined to define the basic geometric relief forms (Krcho, 1973; Krcho, 1991; Dikau, 1989). Each form has a different type of flow. Convex and concave forms in gradient direction have accelerated and slowed flow, respectively, and convex and concave forms in tangential direction have converging and diverging flow, respectively.

Other types of curvatures, such as the principle, mean, or Gauss curvatures as well as curvatures in an arbitrary direction can be computed directly from the interpolation function.

Gradient and curvatures for volumes. Volumes can be modeled by a trivariate interpolation function in the general form of $w = f(x,y,z)$. When this function is differentiate at least up to the 2nd order, the topographic parameters for volumes (3D) can be computed directly from its partial derivatives (Mitasova et al., 1995). First, we introduce simplifying notations for partial derivatives of this function:

$$f_x = \frac{\partial f}{\partial x}, \quad f_y = \frac{\partial f}{\partial y}, \quad f_z = \frac{\partial f}{\partial z}, \tag{B.25}$$

$$f_{xx} = \frac{\partial^2 f}{\partial x^2}, f_{xy} = \frac{\partial^2 f}{\partial x \partial y}, f_{xz} = \frac{\partial^2 f}{\partial x \partial z},$$

$$f_{yy} = \frac{\partial^2 f}{\partial y^2}, f_{yz} = \frac{\partial^2 f}{\partial y \partial z}, f_{zz} = \frac{\partial^2 f}{\partial z^2}.$$

Volume topographic parameters are also derived from differential geometry, using additional independent spatial coordinate (z). Theoretically, such topographic parameters can be derived up to N-dimensional space (see Hofierka, 1997a). For a three-dimensional cartesian space these parameters have the following form:
Size of gradient:

$$|\nabla f| = \sqrt{f_x^2 + f_y^2 + f_z^2} \tag{B.26}$$

Direction of gradient can be defined by two angles.
Horizontal angle A_n:

$$A_n = \arctan\left(\frac{f_y}{f_x}\right) \tag{B.27}$$

and vertical angle B_n:

$$B_n = \arctan\left(\frac{\sqrt{f_x^2 + f_y^2}}{f_z}\right) \tag{B.28}$$

The change of gradient size in its direction has the following form:

$$\frac{\partial |\nabla f|}{\partial n} = \frac{f_x^2 f_{xx} + 2 f_{xz} f_x f_z + 2 f_{xy} f_x f_y + f_y^2 f_{yy} + 2 f_{yz} f_y f_z + f_z^2 f_{zz}}{f_x^2 + f_y^2 + f_z^2} \tag{B.29}$$

When we note principal curvatures in 3D cartesian space as k_1, k_2, k_3, then the Gauss-Kronecker curvature K can by expressed as:

$$K = k_1 * k_2 * k_3 \tag{B.30}$$

The mean curvature M is:

$$M = \frac{k_1 + k_2 + k_3}{3} \tag{B.31}$$

In cartesian system these equations can be expressed as follows:

$$K = \frac{f_{xz}^2 f_{yy} + f_{yz}^2 f_{xx} + f_{xy}^2 f_{zz} - f_{xx} f_{yy} f_{zz} - 2 f_{xy} f_{yz} f_{xz}}{\left(\sqrt{1 + f_x^2 + f_y^2 + f_z^2}\right)^5} \tag{B.32}$$

$$H = \frac{\begin{vmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{vmatrix}}{3\left(1 + f_x^2 + f_y^2 + f_z^2\right)} \tag{B.33}$$

where:

$$h_{11} = \frac{-f_{xx}}{\sqrt{1 + f_x^2 + f_y^2 + f_z^2}} + 2(1 + f_x^2) \tag{B.34}$$

$$h_{12} = h_{21} = \frac{-f_{xy}}{\sqrt{1 + f_x^2 + f_y^2 + f_z^2}} + 2f_x f_y \tag{B.35}$$

$$h_{22} = \frac{-f_{yy}}{\sqrt{1 + f_x^2 + f_y^2 + f_z^2}} + 2(1 + f_y^2) \tag{B.36}$$

$$h_{13} = h_{31} = \frac{-f_{xz}}{\sqrt{1 + f_x^2 + f_y^2 + f_z^2}} + 2f_x f_z \tag{B.37}$$

$$h_{23} = h_{32} = \frac{-f_{yz}}{\sqrt{1 + f_x^2 + f_y^2 + f_z^2}} + 2f_y f_z \tag{B.38}$$

$$h_{33} = \frac{-f_{zz}}{\sqrt{1 + f_x^2 + f_y^2 + f_z^2}} + 2(1 + f_z^2) \tag{B.39}$$

Estimation of partial derivatives. To compute the above described equations for gradients and curvatures we need to estimate first and second order partial derivatives.

In the RST-based modules, partial derivatives of RST functions are used. First, several definitions are introduced

$$\eta = \frac{\varphi}{2} \tag{B.40}$$

$$R'(r_j) = 2\,\frac{1 - e^{-(\eta r_j)^2}}{r_j} \tag{B.41}$$

$$R''(r_j) = 2\,\frac{\left(2(\eta r_j)^2 + 1\right) e^{-(\eta r_j)^2} - 1}{r_j^2} \tag{B.42}$$

Partial derivatives for the bivariate RST basis function can then be expressed as follows:

$$\frac{\partial R(r_j)}{\partial x} = R'(r_j)\frac{(x - x^{[j]})}{r_j}, \qquad l \quad 1,2 \tag{B.43}$$

$$\frac{\partial^2 R(r_j)}{\partial x_1^2} = R''(r_j)\frac{(x - x^{[j]})^2}{r_j^2} + R'(r_j)\frac{(y - y^{[j]})^2}{r_j^3} \tag{B.44}$$

whereas the derivatives, according to y, are found easily from Equation B.44 by exchange of x to y. The mixed derivative is given by

$$\frac{\partial^2 R(r_j)}{\partial x \partial x} = \left[R''(r_j) - \frac{R'(r_j)}{r_j}\right]\frac{(x - x^{[j]})(y - y^{[j]})}{r_j^2} \tag{B.45}$$

These expressions for first and second order derivatives are used for the compulation of slope, aspect and curvatures in the modules `s.surf.rst`, `v.surf.rst` and `r.resamp.rst`. Optionally, the values of these partial derivatives are output by the module `s.surf.rst` when using the flag `-d` .

Partial derivatives for trivariate RST :

$$R'(r_j) = \frac{1}{r_j\sqrt{\pi}} \exp\left[-\left(\frac{\varphi r_j}{2}\right)^2\right] - \frac{1}{\varphi r_j^2} erf\left(\frac{\varphi r_j}{2}\right) \tag{B.46}$$

$$R''(r_j) = \frac{2}{\varphi r_j^3} erf\left(\frac{\varphi r_j}{2}\right) - \sqrt{\pi}\left(\frac{2}{r_j^2} + \frac{\varphi^2}{2}\right) \exp\left[-\left(\frac{\varphi r_j}{2}\right)^2\right] \tag{B.47}$$

In `r.slope.aspect` second order polynomial approximation of a surface defined by given point and its 3x3 neighborhood is used leading to the following equations for the partial derivatives (as used in Horn's formula, Horn, 1981):

$$z(x,y) = a_0 + a_1 x + a_2 y + a_3 xy + a_4 x^2 + a_5 y^2 \tag{B.48}$$

By fitting this polynomial to the 9 grid points (the given point *(x,y)* and its 3 x 3 neighborhood, as shown below), using weighted least squares) we can derive the coefficient of this polynomial as well as its partial derivatives ($f_x = a_1$, $f_y = a_2$, $f_{xx} = 2a_4$, $f_{yy} = 2a_5$, $f_{xy} = a_3$) while using weight $w_k = d_k^{-2}$, $d_k^2 = (x - x_k)^2 + (y - y_k)^2$:

```
x-1,y+1 ------ x,y+1 ------ x+1,y+1
    |              |              |
    |              |              |
    |              |              |
x-1,y -------- x,y ---------- x+1,y
    |              |              |
    |              |              |
    |              |              |
x-1,y-1 ------ x,y-1 ------ x+1,y-1
```

$$f_x = \frac{(z_{x-1,y-1} - z_{x+1,y-1}) + (2z_{x-1,y} - 2z_{x+1,y}) + (z_{x-1,y+1} - z_{x+1,y+1})}{8\Delta x} \tag{B.49}$$

$$f_y = \frac{(z_{x-1,y-1} - z_{x-1,y+1}) + (2z_{x,y-1} - 2z_{x,y+1}) + (z_{x+1,y-1} - z_{x+1,y+1})}{8\Delta y} \tag{B.50}$$

Let us denote $D(x,\delta) = z_{x,y+1} + z_{x,y-1} - 2z_{x,y}$ and $D(\delta,y) = z_{x+1,y} + z_{x-1,y} - 2z_{x,y}$. Then we can write

$$f_{xx} = \frac{D(\delta,y+1) + (4z_{x-1,y} + 4z_{x+1,y} - 8z_{x,y}) + D(\delta,y-1)}{12(\Delta x)^2} \tag{B.51}$$

$$f_{yy} = \frac{D(x-1,\delta) + (4z_{x,y+1} + 4z_{x,y-1} - 8z_{x,y}) + D(x+1,\delta)}{12(\Delta y)^2} \tag{B.52}$$

$$f_{xy} = \frac{(z_{y-1,x-1} - z_{y-1,x+1}) - (z_{y+1,x-1} - z_{y+1,x+1})}{4\Delta x \Delta y} \tag{B.53}$$

where $z_{x,y}$ is the elevation value at row *y* column *x*, Δx is the east-west grid spacing and Δy is the north-south grid spacing (resolution).

B.4. INSOLATION

Equations for computation of solar energy related parameters used in `r.sun` (Hofierka, 1997b, Hofierka and Súri, 2002b, further citations in the manual page of `r.sun`). The clear-sky solar radiation model applied in this module is based on the work undertaken for development of European Solar Radiation Atlas (Scharmer and Greif (eds), 2000, Page et al., 2001, Rigollier et al., 2000).

Solar geometry.

Declination d [rad]:

$$\delta = \arcsin(0.3978 \sin(j - 1.4 + 0.0355 \sin(j - 0.0489))) \tag{B.54}$$

where:

$j = 2\pi day/365.25$ [rad]

Position of the sun in respect to a horizontal plane:

$$sinh_0 = C_{31} \cos T + C_{33} \tag{B.55}$$

$$cosA_0 = \frac{C_{11} \cos T + C_{13}}{\sqrt{(C_{22} \sin T)^2 + (C_{11} \cos T + C_{13})^2}} \tag{B.56}$$

where:

$C_{11} = \sin\varphi \cos\delta$
$C_{13} = -\cos\varphi \sin\delta$
$C_{22} = \cos\delta$
$C_{31} = \cos\varphi \cos\delta$
$C_{33} = \sin\varphi \sin\delta$

Position of the sun in respect to an inclined plane:

$$\sin\delta_{exp} = C'_{31} \cos(T - \lambda') + C'_{33} \tag{B.57}$$

where:

$C'_{31} = \cos\varphi' \cos\delta$
$C'_{33} = \sin\varphi' \sin\delta$
$\sin\varphi' = -\cos\varphi \sin\gamma_N \cos A_N + \sin\varphi \cos\gamma_N$
$\tan\lambda' = -\frac{\sin\gamma_N \sin A_N}{\sin\varphi \sin\gamma_N \cos A_N + \cos\varphi \cos\gamma_N}$

Sunrise/sunset over a horizontal plane:

$$\cos(Th_{r,s}) = -\frac{C_{33}}{C_{31}} \tag{B.58}$$

Sunrise/sunset over an inclined plane:

$$\cos(Tp_{r,s} - \lambda') = -\frac{C'_{33}}{C'_{31}} \tag{B.59}$$

Extraterrestrial irradiance on a plane perpendicular to the solar beam G_0 [W/m^2].

$$G_0 = I_0 \varepsilon \tag{B.60}$$

where:

$\varepsilon = 1 + 0.03344 \cos(j - 0.048869)$

values j and 0.048869 are in radians.

Extraterrestrial irradiance on a horizontal plane G_{0h} [W/m^2].

$$G_{0h} = G_0 \sin h_0 \tag{B.61}$$

Beam irradiance on a horizontal plane B_h [W/m^2].

$$B_h = G_0 e^{(-0.8662\ T_{LK}\ m\ \delta_R(m))} \sin h_0 \tag{B.62}$$

where:

$p/p_0 = e^{(-z/8434.5)}$

$\Delta h_0^{ref} = 0.061359(0.1594 + 1.123 h_0 + 0.065656 h_0^2)/(1 + 28.9344 h_0 + 277.3971 h_0^2)$

$h_0^{ref} = h_0 + \Delta h_0^{ref}$

$m = (p/p_0)/(\sin h_0^{ref} + 0.50572(h_0^{ref} + 6.07995)^{-1.6364})$

where values h_0^{ref} and 6.07995 are in degree

$\delta_R(m) = 1/(6.6296 + 1.7513m - 0.1202m^2 + 0.0065m^3 - 0.00013m^4)$ if m $\leq$ 20

$\delta_R(m) = 1/(10.4 + 0.718m)$ if m > 20

Beam irradiance on an inclined plane B_i [W/m^2].

$$B_i = G_0 e^{(-0.8662\ T_{LK}\ m\ \delta_R(m))} \sin \delta_{exp} \tag{B.63}$$

Diffuse irradiance on a horizontal plane B_h [W/m^2].

$$D_h = G_0 F_d(h_0) Tn(T_{LK}) \tag{B.64}$$

where:

$Tn(T_{LK}) = -0.015843 + 0.030543\ T_{LK} + 0.0003797\ T_{LK}^2$

$F_d(h_0) = A_1 + A_2 \sin h_0 + A_3 \sin^2 h_0$

$A_1' = 0.26463 - 0.061581 T_{LK} + 0.0031408 T_{LK}^2$

$A_1 = 0.0022/Tn(T_{LK})$ if $A_1'\ Tn(T_{LK}) < 0.0022$

$A_1 = A_1'$ if $A_1'\ Tn(T_{LK}) \geq 0.0022$

$A_2 = 2.04020 + 0.018945\ T_{LK} + 0.011161\ T_{LK}^2$

$A_3 = -1.3025 + 0.039231\ T_{LK} + 0.0085079\ T_{LK}^2$

Diffuse irradiance on an inclined plane D_i [W/m^2].

$$D_i = D_h F_x \tag{B.65}$$

where:

if plane is in shade (e.g. $\delta_{exp} < 0$ and $h_0 \geq 0$):

$F_x = F(\gamma_N)$

$F(\gamma_N) = r_i(\gamma_N) + \left(\sin\gamma_N - \gamma_N \cos\gamma_N - \pi \sin^2\left(\frac{\gamma_N}{2}\right)\right) 0.252271$

if plane is sunlit under clear sky:

$if\, h_0 \geq 0.1 rad$:

$F_x = F(\gamma_N)(1 - K_b) + K_b \sin\delta_{exp} / \sin h_0$

$if\, h_0 < 0.1 rad$:

$F_x = F(\gamma_N)(1 - K_b) + K_b \sin\gamma_N \cos A_{LN} / (0.1 - 0.008 h_0)$

$A^*_{LN} = A_0 - A_N$

$A_{LN} = A^*_{LN}$ $\quad if -\pi \leq A^*_{LN} \leq \pi$

$A_{LN} = A^*_{LN} - 2\pi$ $\quad if A^*_{LN} > \pi$

$A_{LN} = A^*_{LN} + 2\pi$ $\quad if A^*_{LN} < -\pi$

$F(\gamma_N) = r_i(\gamma_N) + \left(\sin\gamma_N - \gamma_N \cos\gamma_N - \pi \sin^2\left(\frac{\gamma_N}{2}\right)\right)(0.00263 - 0.712 K_b - 0.6883 K_b^2)$

$K_b = B_h / G_{0h}$

$r_i(\gamma_N) = (1 + \cos\gamma_N)/2$

Diffuse ground reflected irradiance on an inclined plane $\mathbf{R}_i$ [W/m^2].

$$R_i = \rho_g G_h r_g(\gamma_N) \tag{B.66}$$

where:

$r_g(\gamma_N) = (1 - \cos\gamma_N)/2$

$G_h = B_h + D_h$ $\quad$ with G_h in [W/m^2]

Symbols.

- Position of the grid cell (solar plane):

  φ geographical latitude [rad]

  z elevation above sea level [m]

  γ_N slope angle [rad]

  A_N aspect (orientation, azimuth) - angle between the projection of the normal on the horizontal plane and east [rad]

  φ' relative geographical latitude of an inclined plane [rad]

  λ' relative geographical longitude [rad]

- Parameters of the surface (plane):

  ρ_g mean ground albedo

- Date-related parameters:

  day day number 1-365 (366)

  j Julian day number expressed as a day angle [rad]

  T time of computation [decimal hours/rad]

  $Th_{r,s}$ time of sunrise and sunset over the local horizon

  $Tp_{r,s}$ time of sunrise and sunset over the inclined grid cell (plane)

  δ solar declination [rad]

  ε correction of the variation of sun-earth distance from its mean value

- Solar position:

  h_0 solar altitude – an angle between sun and horizon [rad]

  A_0 solar azimuth – an angle between sun and meridian measured from east [rad]

  A_{LN} angle between the vertical plane containing the normal to the surface and vertical plane passing through the center of the solar disc [radl

  δ_{exp} solar incidence angle - an angle between sun and the (inclined) plane [rad]

- Solar radiation:

  I_0 solar_const = 1367 W/m^2

  G_0 extraterrestrial irradiance on a plane perpendicular to the solar beam [W/m^2]

  G_h $G_h = B_h + D_h$ – global solar irradiance on a horizontal plane [W/m^2]

  G_i $G_i = B_i + D_i + R_i$ – global solar irradiance on an inclined plane [W/m^2]

  B_h beam irradiance on a horizontal plane [W/m^2]

  B_i beam irradiance on an inclined plane [W/m^2]

  D_h diffuse irradiance on an horizontal plane [W/m^2]

  D_i diffuse irradiance on an inclined plane [W/m^2]

  R_i diffuse ground reflected irradiance on an inclined plane [W/m^2]

- Parameters of the atmosphere:

  p/p_0 correction of station elevation [-]

  T_{LK} Linke turbidity factor [-]

  T_L corrected Linke turbidity factor ($T_L = 0.8662\ T_{LK}$), see Kasten, 1996)

  m relative optical air mass [-]

  $\delta_R(m)$ Rayleigh optical thickness [-]

- Parameters of the radiation transmission:

  $F_d(h_0)$ diffuse solar elevation function

  $Tn(T_{LK})$ diffuse transmission function

  $F(\gamma_N)$ function accounting for the diffuse sky irradiance distribution

  K_b proportion between beam irradiance and extraterrestrial solar irradiance on a horizontal plane

  $r_i(\gamma_N)$ fraction of the sky dome viewed by an inclined plane [-]

  $r_g(\gamma_N)$ fraction of the ground viewed by an inclined plane [-]

Appendix C
UMN/MapServer sample configuration

This appendix section contains a sample UMN/MapServer definition file and a sample UMN/MapServer HTML template. The example contains read routines for vector data (SHAPE format) and raster data (GeoTIFF and GRASS LOCATION). Please refer to Section 13.4 for further details.

C.1. UMN/MapServer DEFINITION FILE

This sample UMN/MapServer definition file defines the map, projection and coordinates settings.

```
# UMN/MapServer / GRASS:
#              this file contains the map definitions
# This program is Free Software under the GNU GPL (>=2).
# Markus Neteler 2001

# You need:
# * .grassrc5 stored in $DOCUMENT_ROOT
#    (e.g. /usr/local/httpd/htdocs/ ),
#    which may be the HOME of the apache user,
#    containing the entries:
#         GISDBASE: /usr/local/share/grassdata
#         LOCATION_NAME: spearfish
#         MAPSET: PERMANENT
# * a tmp directory: /tmp/mapserver (with Apache Alias)
# * a start file with this URL to the cgi-bin inside:
#   http://localhost/cgi-bin/mapserv?map=\
#              /usr/local/httpd/map-script/mapserver.map
# * the spearfish.html defining the map/html template
# * this mapserver.map containing the map definitions
```

```
NAME GRASS
STATUS ON
SIZE 500 400

# Spearfish
#          W        S        E        N
EXTENT 590000 4914000 609000 4928000
UNITS METERS

# Start of web interface definition
WEB
# reference HTML file:
  TEMPLATE /usr/local/httpd/htdocs/spearfish.html
#temp data absolute path:
  IMAGEPATH/tmp/mapserver/
#temp data relative path:
  IMAGEURL "/tmp/"
# max/min zoom:
  MINSCALE 1500
  MAXSCALE 155000
END # Web

# the small reference map, should get a red zoom box
REFERENCE
  STATUS ON
  IMAGE "/usr/local/httpd/htdocs/spearfish/refmap.png"
  SIZE 187 150
  EXTENT 590000 4914000 609000 4928000
  COLOR 200 200 200
  OUTLINECOLOR 255 0 0
END

# Start of scalebar
SCALEBAR
  IMAGECOLOR 255 225 204
  LABEL
    COLOR 0 0 0
    SIZE tiny
  END
  STYLE 0
  SIZE 100 3
  COLOR 255 0 0
  OUTLINECOLOR 0 0 0
  UNITS KILOMETERS
  INTERVALS 3
  STATUS ON
END
```

```
# Start of legend
LEGEND
  KEYSIZE 18 12
  IMAGECOLOR 255 225 204
  LABEL
    TYPE BITMAP
    SIZE MEDIUM
    COLOR 0 0 87
  END
  STATUS ON
END

# maps, order determines display order: first here is below, following on top

#raster layer available in GeoTIFF (8bit recommened):
# r.out.tiff -t in=elevation,dem out=elevation
# convert -colors 256 elevation.tif elevation2.tif
# mv elevation2.tif elevation.tif
LAYER
  NAME dem
  TYPE RASTER
  STATUS ON
  OFFSITE 0
#switch off map at certain min scale (1:1000):
  MINSCALE 1000
  DATA /usr/local/httpd/htdocs/spearfish/elevation.tif
END

#GRASS raster map directly from location (8bit only):
LAYER
  NAME soils
  TYPE RASTER
  STATUS ON
  DATA "/usr/local/share/grassdata/spearfish/\
       PERMANENT/cellhd/soils"
END # Layer

#vector layer in SHAPE:
#v.out.shape map=roads type=line pref=roads cats=string
LAYER
  NAME roads
  TYPE LINE
  STATUS DEFAULT
  DATA /usr/local/httpd/htdocs/spearfish/roads
  TOLERANCE 5
  LABELITEM CAT_ID
  TEMPLATE /usr/local/httpd/htdocs/spearfish/roads.html
  OFFSITE 0
  CLASS
```

```
    NAME "Road"
    COLOR 80 80 80
    LABEL
      POSITION CC
      SIZE SMALL
      COLOR 0 0 225
    END
  END
END

END # Map File
```

C.2. UMN/MapServer HTML TEMPLATE

This is a sample HTML template file for UMN/MapServer:

```
<HTML>
<HEAD>
<TITLE>GRASS – UMN/MapServer</TITLE>
<META name="Author" content="Markus Neteler 2001 (c) GNU GPL>=2">
</HEAD>
<BODY bgcolor=#FFFFCC>

<FORM method=GET action="/cgi-bin/mapserv" name="mform">
<SCRIPT language="javascript" type="text/javascript">
 FUNCTION fullmap() {
  document.mform.imgext.value="590000 4914000 609000 4928000";
  document.mform.imgxy.value="250.5 200.5";
  document.mform.zoom[1].checked="true";
  document.mform.elements[3].checked="true";
  document.mform.submit();
 }
</SCRIPT>

<CENTER>
<TABLE border=5 cellpadding=10 width=100%>
  <TR><TD>
  <TABLE>
    <TR><TD align="center">
     <INPUT type="hidden" name="mode" value="browse">
     <B>Spearfish (Lawrence), South Dakota</B><BR>
     <INPUT NAME="img" TYPE="image" src="[img]" width=500\
                height=400 bordercolor=#FFFFFF border=3>
    </TD></TR>
    <TR><TD align="left">
    <FONT size=–1 face="arial,helvetica" color="#000000">
    <B>Powered by UMN/MapServer / GRASS</B> </FONT>
```

```
           <img src="[scalebar]" alt="Scale bar"\
                          align="middle">
           <img src="[legend]" alt="Scale bar"\
                          align="middle">
    </TD></TR>
  </TABLE>
  </TD><TD>
  <TABLE width=100%>
    <TR><TD>
    <INPUT TYPE="image" name="ref" src="[ref]" width=187\
        height=150 bordercolor=#0000FF border=1><BR>
    <INPUT type="submit"  value="Refresh/Query"><BR>
    <SELECT multiple name="layer" size=3>
        <option value="soils" [soils_select]> soils map (GRASS)
        <option value="dem" [dem_select]> elevation (GeoTIFF)
        <option value="roads" [roads_select]> roads (SHAPE)
    </SELECT><BR>
    <B>Select map(s) and click:</B><BR>
    <INPUT type="radio" name="mode" value="browse" checked>
              Browse maps<BR>
    <INPUT type="radio" name="mode" value="query" >
              Query road map<BR>
    </TD></TR>
    <TR><TD>

    <INPUT type="radio" name="zoom" value=2 [zoom_2_check]>
              Zoom in (2x)<BR>
    <INPUT type="radio" name="zoom" value=1 [zoom_1_check]>
              Pan<BR>
    <INPUT type="radio" name="zoom" value=-2 [zoom_-2_check]>
              Zoom out (-2x)
    <P>
    <INPUT type="button" value="Full map" onClick="fullmap()">
    <INPUT type="hidden" name="imgxy" value="250.5 200.5">
    <INPUT type="hidden" name="imgext" value="[mapext]">
    <INPUT type="hidden" name="map" value="[map]">
    </P>

    </TD></TR>
  </TABLE>
  </TD></TR>
</TABLE>
</FORM>
</BODY>
</HTML>
```

Index

About the Authors

Markus Neteler, researcher at Centro per la Ricerca Scientifica e Tecnologica (ITC-irst), Interactive Sensory Systems Division, Predictive Models for Biological & Environmental Data Analysis, Trento, Italy.

Helena Mitasova, adjunct associate professor and National Research Council senior research fellow at the Department of Marine, Earth and Atmospheric Sciences, North Carolina State University, Raleigh, U.S.A.