PRACTICAL GEOMETRY

THE MATHEMATICS FOR GEOGRAPHIC INFORMATION SYSTEMS

Andrew U. Frank

Draft manuscript
February 2006 v15

TABLE OF CONTENTS

FOREWORD

I want to tell a story—the story of geographic information systems (GIS). It is a story in small chapters, which are combined to form the complex whole. I believe that the *complexity* of the world results from the composition of small and simple components. The chapters of this book describe the concepts, which combine to produce the complex Geographic Information System. We observe that the same operations are repeatedly used and often implemented in different ways.

I write this text to show that there are mathematical principles behind the construction of Geographic Information Systems and to present these principles as a formal theory. These principles are the same for all applications of GIS. They are based on various parts of mathematics and are likely to be independent of the rapidly changing technology and valid for decades.

Formal treatment is necessary to overcome the terminological confusion in GIS. Geographic Information Science connects with many different, well-established sciences: Computer Science, Information Science, Geography, Geology, Surveying, etc. Each of these has a long and well-established tradition of terminology. There is no single terminology that suits all participating scientists and misunderstandings caused by differences in terminology are rampant in the GIS literature. I select an algebraic treatment—and the corresponding terminology—because it allows integrating in a single coherent picture the treatment of geometric, temporal and descriptive information. Algebra links to category theory, where I believe the definitions of many problems of semantics will ultimately become feasible. I use transformations, mappings, morphism, and functors to stress the relevant structural invariants that must be preserved when representing real-world phenomena in computers.

The *rules for composing* components to construct a complex system is sometimes addressed as design principles or patterns(Alexander, Ishikawa et al. 1977; Gamma, Helm et al. 1995). Unlike computer scientists, I use here methods to

compose functionality based on a mathematical framework: category theory and in particular functors are the guiding principles to identify components and to compose them. Composition is only possible if the components are 'clean' and I will put more effort to establish the foundation than is usual; this will be compensated later when composition is effortless(Wadler 1989);(Frank 1999).

The goal of the book is to cover in a formal way all the theory necessary to understand the core of a Geographic Information System including temporal data necessary to represent change. The focus on formal processing of spatial data results in a number of hotly discussed topics in GI being left out:

- Ontology of geographic data and the Semantics of data;
- Aspect of implementation and performance of algorithms;
- User interface and interaction;
- Approximations, uncertainties, and error.

These limitations are necessary to allow concentrating on what we know well before we start addressing what we do not know and perhaps will never know. I have presented my understanding of these areas in other publications (Frank 1991; Frank 2001; Frank 2003) [ref frank 2004] and expect that a division of the discussion in formal theory of geographic data handling, separated from ontology and semantics, performance and user interfaces leads to a well-structured, effective discussion. The contribution here is restricted to areas where I assume that our current understanding will remain valid for many years.

In a nutshell: this text embraces a constructive approach to GIS Theory: I want to show how a GIS is constructed from a small set of primitive notions and axioms defining them. The analytical approach to consider the applications in the world and deduce the necessary theory will be covered in a separate book with an ontological focus(Frank to appear).

HISTORY OF TEXT

Some parts of the text go back 20 years, to course notes on formal aspects of geographic information systems, which I wrote to support my teaching at the University of Maine (Frank 1985) and the research program that I stated there is still very much the program I follow now (see insert). A comprehensive approach was started in the text I wrote for a graduate course in spring 2000 at the University of California in Santa Barbara. I have

improved and rewritten it for teaching my course "GIS Theory" at the Technical University Vienna.

> [This text] provides a quite generic treatment, suitable for the discussion of any complex information system that deals with a significant part of reality…Often enough a spatial information system is discussed as if it were only a computerized mapping system. … Computer cartography is the subject of a number of courses and a few books have recently appeared on the topic. They discuss how maps can be drawn using a computer and show results that are achieved using typical software packages. Their focus is on the graphical process of map creation and to a lesser degree on map design; very little is said about the source and organization of knowledge about the world that is necessary to draw the map. …
>
> [These] texts on spatial information systems take a radically different approach, trying to encompass the problem of constructing systems that will collect, maintain, and disseminate spatial information. It will be shown that it is clearly beneficial to discuss these problems in context and to understand the interaction among their different components. Using this view, a map is a spatial information system and can be analyzed in these terms, from data collection to map usage. This treatment strives for theoretical correctness and for the formal analysis and specification of a spatial information system. It is based on the observation that many of the problems with present day systems start with shortcuts and seemingly reasonable abbreviations, which later turn out not to be correct and which demand extensive remedial countermeasures. We start with the assumption that "a good theory is the most practical tool" and try to find the principles human cartographers intuitively apply. We try to cast them into a formal language that we can then use to program computerized information systems.
>
> *Excerpt from 1985 course material(Frank 1985)*

Nearly 20 years later, cartography still influences GIS teaching. Cartography has two closely related foci: communication of spatial knowledge and analysis of spatial situations using maps. Waldo Tobler, one of the original members of the 'quantitative revolution in geography' in his Ph.D. thesis (Tobler 1961) gives a framework for analytical cartography based on transformations. His insight to give a mathematical formulation to traditional cartographic methods will be continued here. However, (carto-) graphics and computer analysis should be separated (Frank 1984; Frank 1985) to liberate the GIS from the limitations of the paper map.

TEACHING GIS

I use this text for a second course in GIS in the last year of an undergraduate degree or the first year of graduate studies. The students have attended before an introductory course and used

some commercial GIS software to work on example problems, which lead them to a basic understanding of the typical GIS applications.

The "GIS Theory" course is a three credit course of 15 weeks duration, where one part of this book is presented per week. Engineering students have covered before discrete mathematics course in linear algebra and vector and matrix operations. The material in these chapters need only be reviewed to connect it to GIS, but the text is self-contained and not dependent on any special math requirements beyond high school.

I do not know of another textbook intended for a second, rigorous GIScience course. The question what to include and how to structure is not less difficult then for the introductory course where several text books exist with different content. During the 1980s, I divided my teaching in a course on the storage and retrieval of geographic data and another one covering geometric aspects of geographic data processing. This division became obsolete as the integration of databases and graphical data processing into mainstream progressed.

For a post-graduate course we asked users 1993 what to include and how to structure the material. The result were three volumes: theory, implementation, and usage(Unwin 1990; Kemp 1993; Kemp, Kuhn et al. 1993). Later, the focus moved towards understanding what the input data meant and how to interpret the results produced by the GIS: spatial cognition and ontology {Frank, 1995 #348; Frank, 1995 #349; Frank, 1995 #350; Frank, 1997 #175}. Students—especially students in an engineering curriculum—had difficulties to grasp the questions of semantics, data quality, etc. while at the same time learning the technical aspects of Geographic Information Systems. An attractive course outline based on different aspects of cognitive space (Couclelis and Gale 1986) did not include enough of the basic knowledge necessary for use; I abandoned it as yet another attractive but not pedagogically suitable guideline.

The approach followed here is novel as it concentrates on the part of the GIS theory we can explain with formal (mathematical) methods. It should appeal to engineering and computer science students, but also to students in a graduate program in geography with a bend to formal sciences. Our knowledge of Geographic Information Science has sufficiently

increased during the past years to cover the fundamental aspects in a formal way.

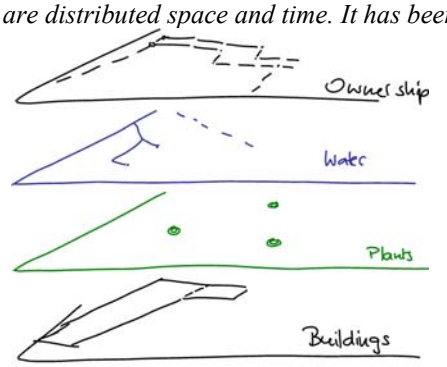
The translation of the formulae to code and to demonstrate that this is sufficient for a *model* of a GIS was done in parallel to the writing of this text. It demonstrated that this foundation is comprehensive and no major holes are left. A number of typical GIS application questions can be solved with the theory presented here. Nevertheless, I invite students, fellow teachers, and researchers in GI Science to send me suggestions for topics I left out and inform me of errors in the presentation.

ACKNOWLEDGEMENT

A very large number of people have contributed in one or the other form to my understanding of GIS.

PART ONE INTRODUCTION

A GIS integrates data describing different aspects of the world and how they are distributed space and time. It has been visualized as a "layered cake"



(Figure 1): different aspects of reality are represented as layers, which are coordinated. GIS software must facilitate exploitation of thematic data with respect to location and time. A GIS contains functions to manipulated geographic data and is separated from other programs that treat text, photographs, etc., but integration of geographic data with other data in a single environment has started.

Geographic Information Systems today are computerized systems, which treat geographic data. Geographic data processing is the processing of data that has a relation to the world (see chapter 3).

This short first part of the book surveys the territory of Geographic Information Systems. It explains my understanding of what a Geographic Information System is and which major applications I think of. A brief history of GIS over the past 25 years should give some historic perspective.

The second chapter gives an overview of the text and how it is structured. It details also what is left out and justifies the focus.

The third chapter describes the GIS as a repository of a description of the real world. It establishes terminology and gives the frame for the rest of the book and explains the generally used terms *system*, *model,* etc.

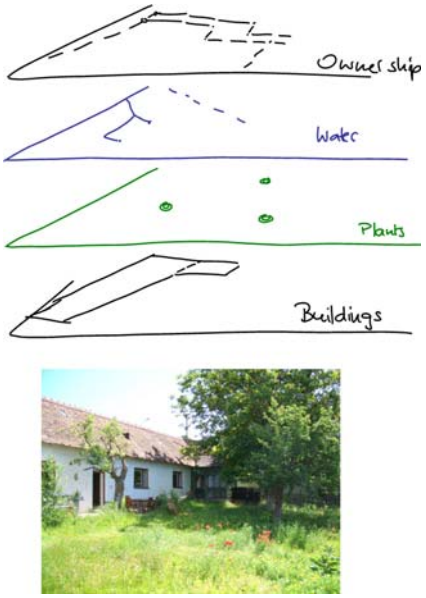


Figure 1: The layered cake: GIS brings together data related to the same location in space

Chapter 1 WHAT IS A GEOGRAPHIC INFORMATION SYSTEM?

Geographic Information Systems—commonly abbreviated as GIS—have evolved in the past 35 years from systems for specialists to produce maps with computers to programs that ordinary people use to solve ordinary problems: GIS is used for planning of urban development, make thematic maps for newspaper articles and help with the navigation in our cars.

This chapter lists the different strands of evolution that lead to present day GIS and reviews some application areas: each discipline and application area has contributed its own conceptual framework and terminology, influences that are still felt today.

1. ORIGINS OF GEOGRAPHIC INFORMATION SYSTEMS

The roots of Geographic Information Systems can be seen in different developments that all introduce electronic data processing to some parts of geographic practice. Several, more or less comprehensive descriptions of the history of GIS exist (Tomlinson, Calkins et al. 1976; Rhind 1991; Rhind 1991; Kemp, Kuhn et al. 1993; Frank 1995; Mark 1997). David Rhind gives a graphical representation of the family tree of today's systems(Maguire, Goodchild et al. 1991).

The pioneering work of Roger Tomlinson introduced electronic data processing to the gigantic task of managing the natural resources of Canada. He coined the name of Canadian Geographic Information System in 1967(Tomlinson 1984). The Canadian GIS maintained maps showing an inventory of the natural resources of the Canadian territory—a task that was beyond what could be achieved with manual cartography.

At about the same time, researchers at the Harvard Graphics Lab computerized the classical method of overlaying maps for cartographic analysis used in urban and rural planning with translucent paper sheets (McHarg 1969; Steiner and Gilgen 1984). The computerized system can combine more layers than cartographers using paper maps and it can integrate data from different sources and in different scales(Chrisman, Dougenik et

al. 1992). For example, different administrative boundaries and census data can be combined with topographic maps.

The researchers at the Harvard Graphics Lab moved on to the commercial world. Jack Dangermond founded the Environmental Systems Research Institute (ESRI) in 1969. It provided geographic data processing and analysis services. In 1980 they offered for sale their programs under the name of ArcInfo, geared primarily to planners. David Sinton (1978) left the Harvard Graphics Lab to join Intergraph, which—together with Synercom—was one of the leading companies to produce GIS for public utilities.

The US Bureau of the Census investigated the use of computers to produce the maps to organize the collection of census data in the field. They had mathematically trained staff, including James Corbett(Corbett 1975), Marvin White (White 1979; White and Griffin 1979) and later Alan Saalfeld, who made early theoretical contributions, which lead to the widely used, standardized, topological Dual Independent Map Encoding (DIME)(Corbett 1975).

The utility of the electronic computer to automate the labor intensive tasks of cartography was recognized early on. The Experimental Cartographic Unit of the Ordnance Survey UK focused on the computer-assisted production of high-quality printed maps(Rhind 1971; Tobler and Wineberg 1971). Using the computer to produce topographic maps, to construct thematic maps, and to maintain the large collections of relatively simple line graphs for public utility and real estate cadastre became possible(Messmer 1984).

In Germany, a group working on the conversion of cadastral maps to computer databases (Automatisierung der Liegenschaftskarte ALK) was active since 1970(Neumann 1978). This project is not completed yet and holds most likely the record for the longest running GIS project ever! In 1973 the preparation for the conversion of the Austrian cadastre started and the conversion was completed in 1984(Hrbek 1993). Public utilities reported successful and cost effective use of early GIS that integrated computer drawn maps with the corresponding administrative databases(Frank 1988).

These different applications lead to different communities of users and developers, with limited communication. The series of

AutoCarto conferences began in 1974.In 1978 the two first "general" GIS conferences were organized in the USA by the Harvard Graphics Lab (Dutton 1978) and in Germany by the Geodesists of the Technical University of Darmstadt(Eichhorn 1979). Application oriented and regional conferences emerged in the USA and Europe during the 1980s. In 1984 the Spatial Data Handling Conference (SDH)(Marble 1984) started {Marble, 1984 #1724; Blakemore, 1986 #10384}.

The US National Center for Geographic Information and Analysis was the result from a national competition(Abler 1987; Abler 1987; NCGIA 1989); it is a consortium of the University of California Santa Barbara, the New York State University Buffalo, and the University of Maine—two geography and a surveying engineering department—connected by a common research agenda (NCGIA 1989a). It organized numerous research meetings, called specialist meetings, to document the state of the art and to identify research questions [ncgia publication list]. Researchers associated with the NCGIA initiated several successful series of bi-annual conferences:

- SSD for large spatial databases in 1989(Buchmann, Günther et al. 1990); this conference takes a Computer Science perspective and discusses spatial access methods, query processing, etc. for geographic information. It is now called SSTD for Symposium on Spatial and Temporal Databases.
- COSIT for Spatial Information Theory (COSIT) in 1992(Frank, Campari et al. 1992; Frank and Campari 1993), collecting contributions from an interdisciplinary range of disciplines: human geography, cognitive science, mathematics, computer science, etc.
- GI Science conference is held bi-annually and addresses the whole field of Geographic Information Science(Caschetta 2000).

2. APPLICATION AREAS FOR GIS

Humans live in the spatial environment; all human activities require space and management of space—from real estate markets to urban planning(Abler, Adams et al. 1971). Space controls aspects of human interaction(Hillier and Hanson 1984; Hillier 1999). Humans are navigating in space and require information about the location of desirable locations and the path to them. All human activities require space. It is estimated that

80% of all data contains some relation to space—which indicates how prevalent spatial aspects in information handling are and that nearly all decisions are influenced by spatial information or the outcome of the decision has spatial effects.

Application areas for GIS are many and a systematic classification difficult. In the abstract, three roles for a GIS are sometimes differentiated:

- *maintain an inventory* of some objects in space;
- *analyses* of spatial situations, mostly for urban and regional planning; and
- *mapping* of geographic data.

The *management* of resources located in space is of universal importance. GIS help to manage the environment, forest and mineral resources. Decision support systems provide tools for the analysis and assessment of the impact of planned actions. Improvement in the management of land is the result of using cadastre, facilities management systems, forest information systems, etc. Improvement in land management contributes to the economic development of a country.

GIS are used in *urban and regional planning*. Computers produce comprehensive graphical presentations of the current situation and the systematic evaluation of options in the planning process and visualize different scenarios.

The *maintenance* of large *map collections*—topographic maps produced by National Mapping Agencies and the collection of maps showing the lines of a public utility, e.g., the gas or water lines buried in the streets of a city—were the dominant applications in the 1980s.

The combination of cartographic—mostly graphical—data with descriptive data permits *analytical* use of the data: one can identify objects and regions based on some criteria. For example, the water authority can identify all water mains, constructed from a troublesome material, and thus produce a plan for preventive maintenance of its water distribution network. This reduces interruption of services to customer and also cost for repair. Similar analytical functions help the forest manager to identify the forest stands to cut during the next years.

Geographic information is used in *business*. For example, the decision to locate a new multiplex cinema or the selection of bank branch offices that should be closed is both dependent on

the spatial distribution of potential clients around the locations. The analytical functions in a GIS produce the information on which rational decision can be based.

The different applications, but also the different disciplines contributing historically to GI Science used different concepts of GIS. The graphical paradigm of cartography—a truthful graphical representation of the real world—remains influential for GIS and GI Science (MacEachren 1995) and clashes with the paradigm of knowledge representation that dominates administration, database design, and decision support systems, which all build conceptual models of reality(Kent 1978; Lockemann and Mayr 1978). Today, the limitation is the lack of tools for the integration of temporal data(Frank 1998).

In general GIS are used to make decisions: users retrieve information that they think is relevant for their decision and use it to improve their decision. This is, incidentally, the only use one can make of information.

The only use of information is to improve decisions.

REVIEW QUESTIONS

- What were the first functions GIS precursors fulfilled?
- When was the first GIS (with this name) constructed? For what purpose? By whom?
- How can the application of GIS be classified in three large groups?
- What are the primary application areas of GIS? (Name five)
- What is the difference between GIS and cartography?
- Describe the evolution of GIS.
- Do you believe that 80% of all decisions we make involve spatial information? Give examples for decisions that are not influenced by spatial information and the outcomes do not influence space.

Chapter 2 FOCUS OF GIS THEORY: OVERVIEW OF TEXT

How to understand GIS? How can we explain software that took hundreds of person-years to write and have manuals many hundred pages long? The commercial GIS courses train people on how to use a GIS product and explain GIS concepts from the perspective of a product(ESRI 1993). An academic course must be independent of products and focus on the core of a GIS.

1. WHAT IS 'GEOGRAPHIC INFORMATION SYSTEMS THEORY'

In general it is assumed that Geographic Information Systems are a tool and do not have a theory. Many have pointed out that there is no science of hammers and similar tools and have suggested a science of Geographic Information Science (Goodchild 1990; Goodchild 1992; Goodchild, Egenhofer et al. 1999) and denied the existence of a theory behind GIS.

A number of applied sciences—what may be called 'topical' sciences—work on problems that connect to space; for example population studies or hydrographic research. Geography concentrates not on the 'topical' aspects of an application, but on the general understanding of processes in space(Abler, Adams et al. 1971). Geographic Information Science is investigating the questions of treatment of geographic information in general—it is an abstraction from different parts of geography and related sciences. Geographic Information Science investigates commonality between the different methods to treat geographic information and to establish some coherent body of knowledge as a common foundation for geographic analysis.

Geographic Information Systems Theory concentrates on the representation and treatment of description of geographic facts and processes. It is the science of Geographic Information Systems, which are the technical systems with which geographic information is treated. GIS are used in most spatial sciences. GI science is a substantial subfield of geography. GIS theory is a subfield of GI science, founded on mathematics and computer science, with contributions from geodesy and measurement sciences(Krantz, Luce et al. 1971).

GIS theory is to GIS what physics to hammers is!

The theory of GIS is intentionally a theory of the tool, a theory of the hammer so to speak. There exists, despite the aforementioned opinions to the contrary, a theory of hammers: it

is physics, in particular mechanics, which deals with movement of masses, levers, impulse transfer from one mass to another upon impact, etc. The theory of GIS, presented here underlies the implementation of the currently available commercial GIS programs. To overcome two of the most obvious shortcomings of today's commercial GIS software, the GIS theory must show how different representations of space can be integrated and contain methods to deal with temporal aspects—including changing values, processes, etc. This will be explained here.

2. TARGET OF THIS BOOK

The purpose of this book is to describe methods that are used in Geographic Information System software. It stresses the concepts that remain likely invariant under the changes that are brought on by technology—from ever faster CPU to the revolution of the World Wide Web. It seems futile to teach students facts that are immediately superseded by the rapid advances of technology, only formal theories do not change.

Mathematical truth does not change with the years!

The description concentrates on what the functions in a GIS do, not how they are implemented. I think it is necessary to understand the basic algorithm before one starts to decide on its implementation. Implementations involve trade-offs depending on the particulars of the application and the current state of technology(Frank 1991). Much of what is currently maintained as 'well-known' rules in GIS software design depends probably more on past technology than we are aware of. Some of these 'well-known' rules may be patently wrong today, made obsolete by new technology and its different performance characteristics and the relevance of others for tomorrow's implementation doubtful.

Identifying the concepts—independent of application and technology—helps to separate what is logically necessary and what is baggage that was once necessary but can be shed today to construct lean systems. The novel aspect of this treatment is the focus on the construction of a *formal* theory of GIS software. The integration of time into GIS is a first demonstration of its usefulness.

Excluded:
- Ontology and semantics of data
- User interface
- Errors and uncertainty in the data
- Performance

A theory of geographic data processing can be developed if one is ready to leave out areas where we have only limited knowledge:

- Ontology and Semantics: All aspects of the meaning and use of the data in real world are excluded(Frank 2001; Frank 2003). We assume that data with fixed and known interpretation is fed into the system and the results are interpreted in the same context, the details of which are left out. This excludes all considerations of what the data means, how it relates to the reality it represents and how treatment in computer systems of spatial information corresponds to the human cognitive abilities.
- User interface: The communication between user and GIS is necessary to effective use of GIS technology. It is closely linked to questions of semantics and for the same reasons excluded here(Frank 1982; Egenhofer and Frank 1992).
- Errors and uncertain data: Current GIS deal well only with data that is precisely known. Real world situations are neither well-defined nor precisely known. Understanding spatial data processing in the precise case contributes to handle imprecise and erroneous data later (Goodchild and Gopal 1990; Burrough and Frank 1996; Goodchild and Jeansoulin 1998; Heuvelink 1998; Shi, Fisher et al. 2002; Frank and Grum 2004; Frank and Grum 2004; Pontikakis and Frank 2004).
- Performance: Technology advances affects primarily how fast operations perform(Frank 1991). Transformation to convert a naïve algorithm to a more performing one are studied in computer science and is left here to the implementer(Bird and de Moor 1997).

Some will argue that the topics excluded are all the really interesting and difficult ones—and I readily agree. These excluded topics are difficult because they appear currently as *ill posed* problems, not amenable in the form they are presented to formal treatment. There are no criteria available to determine the 'best ontology', to compare two implementations or to judge the effectiveness of a user-interface. The topics excluded are those that link the formal treatment of geographic data to its use, to the give and take of the world, to politics and power. In this book I try to cover all the areas that I see fit today for formal treatment. I hope to provide a firm ground for future approaches to some of the problems excluded here. Without this clear separation, we taint the description of the things we presently understand with our ignorance in other areas.

GIS Theory is similar in what it covers to standardization.

It is interesting to note that the focus used here—excluding application areas, performance or the specifics of interaction—is similar to the point of view taken by current standardization efforts, especially in the Open GIS Consortium (OGC 2000) and the ISO TC 211(ISO 2004). Standards—if understood correctly—must concentrate on fixing what should be done and leave the different vendors free to select how they want to achieve it.

3. FORMAL APPROACH

Each part of mathematics comes with its own terminology. To integrate them in a single system, a common notation is necessary. This was the overall purpose of the monumental effort by Whitehead and Russell writing the Principia Mathematica(1910-1913), but also of the French project Bourbaki. These two groups saw in *set theory* the foundation and attempted to build all other parts of mathematics on this base. I follow here the lead of theoretical computer science using algebra (Goguen, Thatcher et al. 1975) and category theory(Asperti and Longo 1991). Standard engineering mathematics, mostly calculus, is useful, but discrete mathematics and algebra (Mac Lane and Birkhoff 1991) are required for GIS Theory(Ehrig and Mahr 1985; Ehrich, Gogolla et al. 1989). Category theory provides a general framework to integrate different parts of mathematics, for example set theory and analysis, linear algebra, topology but also graphs, formal languages, and the theory of finite automata (see xx). To make the text self-contained, these foundations are reviewed as far as they are used.

The focus of the book is on the concepts and not the implementation, thus a mathematical notation using the framework of category theory (Pitt 1985; Barr and Wells 1990; Herring, Egenhofer et al. 1990; Asperti and Longo 1991; Walters 1991; Pierce 1993; Frank 1999) is usually sufficient. In a few rare exceptions, programming languages must be borrowed, where I prefer the Functional Programming Language(Backus 1978), using the syntax and semantics of Haskell (Hudak, Peterson et al. 1997; Peterson, Hammond et al. 1997; Bird 1998) and the imperative language Pascal(Jensen and Wirth 1975). No knowledge of these languages is assumed.

4. STRUCTURE OF THE BOOK

The text consists of eleven parts. This introduction explains the relation between the world, GIS and GI Theory. The second part sees the GIS as a repository of a description of the world. It introduces the formal languages and methods to build theories and uses them to describe measurements.

The third part covers continuous space and time. It introduces time points and vectors to represent points in space, with the pertinent operations, and develops a general theory of spatial transformations.

Part four then constructs functions that operate on map layers (like figure 1.1xx) or time series from functions relating properties of points.

With the fifth part we enter the world of objects in space and how descriptions are stored in a database. Sharing of data among many programs leads to the centralization of data where it can be accessed with standardized functions. To maintain this data consistent for long periods of time, despite many concurrent users, requires special approaches.

The sixth part concentrates on infinite geometric objects: infinite lines, planes, etc., the relations between them and operations applicable to them. It uses projective geometry to give a most general and dimension independent description.

The seventh part focuses on geometric objects with boundaries: line segments, triangles, etc. The simplest geometric objects for each dimension are called simplices and operations applicable to them, again independent of dimension, are given.

The eight part looks at cartographic lines and how they structure space. It uses algebraic or combinatorial topology to discuss spatial subdivisions and the operations that leave the Euler formula for polyhedron invariant.

The ninth part discusses aggregates of lines, which form graphs. Practically are graphs that represent street or stream networks.

The tenth part specializes to a special form of subdivision, namely triangulation. It shows how they are constructed and used for the representation of Digital Terrain Models, or for the determination of service areas around service points using the Voronoï diagram, which is the dual of the Delaunay

triangulation. It also gives a method to compute intersections between arbitrary geometric figures.

Part eleven covers temporal data for moving objects. It demonstrates that the framework is general enough to treat moving objects. Spatio-temporal database with the necessary two time perspectives are constructed as databases with changing content.

Each part consists of short chapters, which are generally motivated by a practical example of geographic data processing. These examples connect the theory to concrete applications of GIS. The summary at the end of each chapter indicates what concepts to retain and links them to the following chapters. Each chapter contains also a list of review questions.

REVIEW QUESTIONS

- What is the focus of GIS Theory? Compare with GIScience.
- Why is the content of GIS Theory similar to the efforts to standardize GIS functionality to achieve interoperability between GIS managed by software from different vendors?

Chapter 3 INFORMATION SYSTEMS

> The *Hitchhikers Guide to the Galaxy* is an indispensable companion ... In case of major discrepancy it is always reality that's got it wrong. (Adams 2002, 172)

In order to understanding the world (Figure 2a), we construct representations of it, for example as a topographic map (Figure 2b). In this chapter, the relations between reality and representations are explored. We will see that Information Systems are models of reality such that a correspondence exists for some operations and objects in the world and their representation in the model; we say that the model (i.e., the topographic map) has an *interpretation*. The interpretation for a map is given as natural language terms in the map legend. A GIS is useful as far as it is a true model, which means that there is a mapping between reality and information system that preserves structure.

Morphism: a structure preserving mapping.

This chapter gives intuitive definition for often used but seldom defined terms like system, model, and data. It shows how they relate and how I intend to use them in this book.

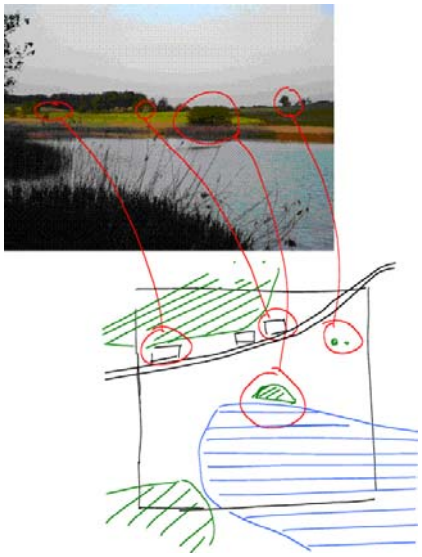


Figure 2 (a) Reality—a landscape near Geras with (b) the corresponding map

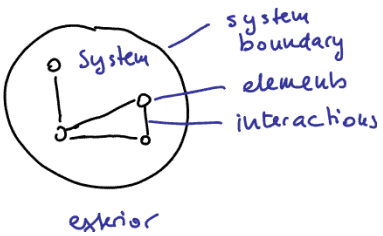


Figure 3: A system, its boundary, its elements and the interaction between them.

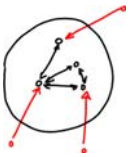
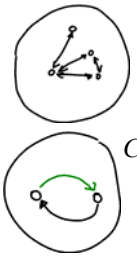


Figure 4: Open System



Closed System

Figure 6: Homeostatic system with feedback-loop

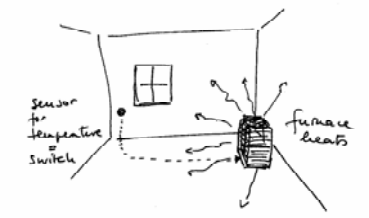


Figure 7: Self stabilizing heating system with feedback loop

A system is a conceptualization, not a reality. Different systems can be identified at the same location.

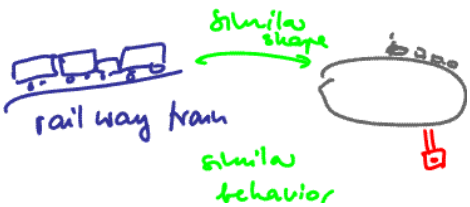


Figure 8: Railway and model

1. What Is a System?

The word *system* is often used, not always with a clear understanding what is meant. General systems theory (Bertalanffy 1973) emerged from biology and considers a system as a delimited collection of interacting parts (Figure 3). The system has a *boundary*. *Closed systems* have no exchange with their environment; all interactions are among elements within the system boundary (Figure 5). *Open systems* interact with elements outside the system boundary (Figure 4). Systems that can maintain their internal state constant are called homeostatic (Figure 6). Figure 7 gives the familiar heating control system as an example for a feedback loop to stabilize the temperature in a room.

To consider something as a system, it is necessary to give its boundary and its interaction with the environment, the elements are identified and their interactions described. Interactions between elements can be material or informational. Systems are often analyzed in a hierarchical fashion. Starting with a coarse decomposition, it is possible to decompose and study each part, e.g., the thermostat (in Figure 7) may be considered again a system with interacting parts.

2. Model

A model *represents* a system that is a part of reality. The model railway I played with as a boy (Figure 8) represents a real train that I was not allowed to play with.

Models are used for the study and prediction of the behavior of a system without affecting the original; they are necessary, whenever experimenting with the real system is impossible, hazardous, or expensive. Scientists and engineers build formal models of systems in which they are interested and work with the model instead of the real system. We do not want to build bridges with a trial and error method—some hold up and some crumble—nor do we want to test the effects of major accidents in nuclear power plants! We build models – computational or reduced scale – and use theory to predict the outcome of operations, without the risk or expenses of real systems.

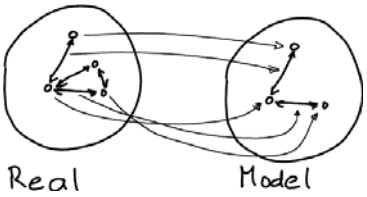


Figure 9: The connection between real system and model

A model is an 'image' of a part of reality (Figure 9). The appropriateness of the model is determined by the usefulness of information it provides about the part of reality modeled. What elements and what relations from the real world should be included in the model? This is a question of how to limit the system that is modeled. There are many trade-offs to consider in choosing a model. Making a model more complete by adding detail is not necessarily make it more accurate. The inclusion of more detail makes the model more difficult to use or introduces too many uncertainties, such that the results are less reliable than what we achieve with a simpler model.

Many models are reduced scale artifacts that are similar in shape and have similar behavior. We call these *analog models* (Figure 8). Maps are graphical models of reality (Figure 2b). *Computational models* are constructed with symbols manipulated according to rules in a computer (see Part 2), simulating the behavior of the system. The observations in the real world must be in a known relationship to the representations in the model as shown in Figure 2. The mapping from a real system onto a formal system is what makes the model useful. Mathematically we can see a situation similar to a homomorphism (see later chapter 5), which is a mapping that preserves (algebraic) structure.

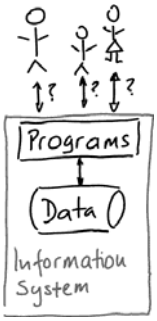


Figure 10: An information system contains data and programs to process the data; it answers questions

3. INFORMATION SYSTEMS

An information system is one that has the main task to produce information (Figure 10); other aspects of the physical data processing machinery, e.g., the consumption of energy and the production of heat are disregarded. Information systems contain data and programs that are used to answer queries of human users. An information system may not connect directly to the data but to other information systems to obtain the answers on behalf of its users (Figure 11); this gives an easy approach to separate user interface issues from the management of the data.

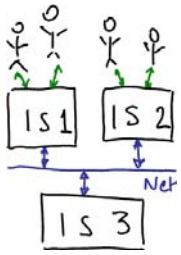


Figure 11: Users connect indirectly to an information system through the net

4. GEOGRAPHIC INFORMATION SYSTEM

A geographic information system is an information system where data is related to physical (geographic) space and operations exploit the data with respect to the spatial location of the objects represented. For each GIS one must determine the region of the world it is describing; this amount to a definition of

a part of the world as a system, which is of interest and represented in the GIS. The GIS with the data is, of course, also a separate system, consisting of electronic equipment, programs, procedures, etc.).

Not every collection of data with some geographic references makes a GIS: there must be analytical functions programmed, which allow users to analyze the data with respect to spatial location. For example, a telephone directory is not a GIS. It contains addresses but does not allow spatial questions—one cannot ask "What is the closest police station to this phone number?" A GIS geocodes the addresses and then use the coordinates to answer this and many similar questions.

5. DATA AND INFORMATION

5.1 INFORMATION

The term *information* will be reserved for contributions to the users' mental models. Information is only that which humans perceive and add to their mental models, and is not the raw material, that is, data or documents, from which they get this information. Only signs that can be perceived and interpreted by humans should be called information. Information is relevant only as it is used to make decisions that lead to actions.

Information is an answer to a human's question.

This definition excludes a number of things that are often considered information. For example, a telephone directory is not, by itself, information, as humans do not ordinarily read and comprehend it. We rather use it as an information system for extracting the information we need when we want to call someone.

5.2 DATA AND DOCUMENTS

The word *data* will be used for symbols represented in a formal language and assumed to have a fixed and known interpretation. Data are in a form accessible to computer hardware, e.g., encoded and stored on media that are accessible to computers.

The word *document* denotes information recorded in a natural language. Documents require a human to interpret their content. Examples are the registry of deeds, libraries, and maps. Documents are not information unless they are read by a human user, but it is also not data, as it cannot be manipulated within a formal model.

Data = Signs (symbols) in a formal language.
Information = Material for constructing mental models.
Document = Signs in a natural language that needs human interpretation.

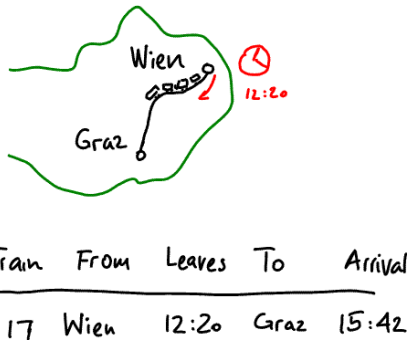


Figure 12: Train information system as a model

Data is in the formal realm—linked by the interpretation to the physical reality—and is thus amenable to mathematical rigor.

6. Information Systems as Model

An information system is useful if the information in it corresponds to the situation in the real world, if it is a (computational) model of a part of reality. If we ask the information desk of the Austrian Railways Company "what is the next train from Vienna to Graz" and get the information that one is leaving at 12:20 p.m. at Wien-Südbahnhof we expect this information to correspond to the real world event when the train leaves the station and we will be ready at the platform a few minutes earlier. The train information system accessible at www.oebb.at is a model of some aspects of the Austrian railway system.

Correct data = Interpretation of the data corresponds to reality.

For the information system to inform about the world there must be a defined relationship between the data and the objects in reality. We say that information is correct, if it follows the conventional, agreed interpretation of the data(Kent 1978). The mapping between data and real objects must preserve the structure that exists between the objects: the connection between 'train to Graz' and '12:20 pm' must be the same as the relation between the train and its time of departure. It is not sufficient that we model the elements of the system, but we have also to model the relations between the elements (Figure 9). We will use the term *interpretation* for this relation between the features in the world as we experience them and the things in a computer program. The computer program with a known interpretation is a model—similar to a small mechanical model used to see how a machine works (Figure 8).

Interpretation: a convention to connect symbols to real world phenomena.

In mathematics this mapping is described as *morphism*: a structure preserving mapping. The real world objects and their connections must have the same structure than the corresponding data objects and the links between them. Algebra gives a succinct definition of structure (see Part 2, chapter 6). Asking about the path of the train from Wien to Graz must result in the information about the stations that the train will call at. If this correspondence does not exist the information obtained from the information system is not useful; a system that informs us that the train leaves at 12:20, but arriving at the station at 12:10 we just see the train pulling out of the station is useless, because the

information is *not correct*, not following the conventional interpretation of what '12:20' etc. means..

Morphism: a mapping that preserves algebraic structure.

6.1 CORRECTNESS OF AN INFORMATION SYSTEM

Users of information systems assume implicitly that they gain the same information, i.e., the same mental models, by consulting the information system, as they would by going out and gathering the information themselves through direct perception of reality (Figure 13).

- You assume that the telephone number you receive from directory assistance is the same you would obtain by going to a person's home and reading it from their phone.
- The tax assessors consulting their lists of parcels and frontages assume that the results are the same as if they went out and measured for themselves.

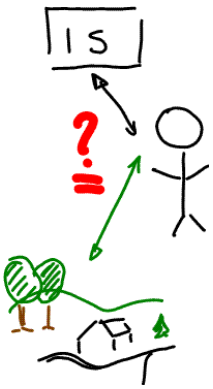


Figure 13: The information system provides the same information than investigating reality

Data stored in the database of an information system must be *correct* to be useful, that is, faithful representation of the structure in reality. A computerized system cannot, by itself, guarantee factual correctness; it has no way of going out and checking that the grass is green, that the moon is not made of cheese, or that the house at 16 Maple Street has fourteen windows. To assert correctness, we have to leave the information system (the formal model) and compare it with reality (Figure 14).

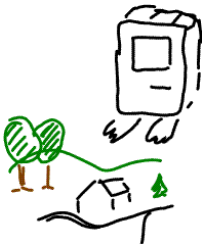


Figure 14: The Banana Jr. computer inspects correctness of the data in the world

Within the information system, formal checks can only assert the weaker notion of *consistency*, which means that the database must be free of internal contradiction (see chapter 18). For instance, the database should never contain information at the same time that the building at 16 Maple Street has 8 and 14 windows; if we find information that the train to Graz leaves Südbahnhof at 12:15 and 12:20 we are confused and wonder if there are really two different trains or rather one of the two times is wrong – but which one?

6.2 AN INFORMATION SYSTEM AS A FORMAL MODEL

The abstract view of an information system retained sees it as a system of symbols together with an interpretation that links the formal symbols to reality (Figure 2). A computerized information system is a *formal model* of a part of reality. The formal system, executed by the computer, operates on symbols that have an interpretation in the model perceived by people.

Information systems are useful if the mapping between symbols and real objects preserve this structure.

All operations of computers are symbol manipulation. Human users tend to interpret computer operations differently, for example as a numerical computation, or even as a complex operation like booking an airline passage. The internal operation of a computer is never anything more than a manipulation of symbols according to formal rules laid down in programs. Computers represent symbols internally in bit patterns. Hardware and software operations are built to manipulate those patterns in a way consistent with our understanding of arithmetic or logical operations.

7. SUMMARY

A GIS is a representation of a part of reality. The interpretation of the symbols stored and treated in the GIS link the model to a part of reality. The treatment of the symbols in a useful information system corresponds to the part of reality represented. In the remainder of this book the rules for symbol manipulation that preserve the intended geographic interpretation will be discussed.

REVIEW QUESTIONS

- What is the definition of an information system; what is specific about a geographic information system?
- In what sense do computers know about a train leaving?
- What is the (only) use of information?
- Why is an ordinary phone directory an (non-automated) information system, but not a GIS?

  What is the difference between information and data?
- What is the difference between correctness and consistency?
- What is an interpretation of a model?
- What is a structure preserving mapping? What is meant by structure in this context?

PART TWO GIS AS A REPOSITORY OF A DESCRIPTION OF THE WORLD

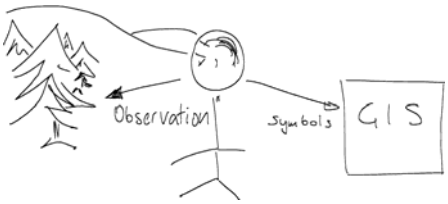


Figure 16: Observations of the world are put into the GIS

Observations of the outside world are stored in the information system. Observations are the linkage between the real world in which we and the GIS operate (Figure 16). A description of the GIS must start with observations, measurements and how they are represented in the GIS. This part introduces methods to construct symbols to represent the result of observations and to manipulate these as well as methods to measure information content. Observation processes *per se* are discussed in the ontology and are not investigated in this book.

Terminology:
Observation (processes) results in measurements.

In general, I will use the term *observation* for the process that connects the real world with the realm of information; *measurement* will be used for the representation of the result of an observation, measurements are often, but not always expressed on a numerical scale.

The previous chapter reviewed the concept of an information system, which is a system that stores and transforms symbols. Symbols represent the outside world in an information system. In order to describe the theory of GIS, two issues must be addressed:

- the representation of values obtained from observation of reality,
- the rules for transformations of representation (i.e., data processing).

The first chapter introduces formal languages to produce representations for the results of the observations in an information system. First order predicate calculus is an example of a formal language, widely used for the description of information systems(Gallaire 1981; Gallaire, Minker et al. 1984).

The second chapter reviews algebras and categories, which seem more apt to represent processes that change the world.

Category theory is considered as the theoretical foundation of computation(Asperti and Longo 1991).

The third chapter discusses what operations can be applied to measurements. It starts with Steven's classical scales of measurement (Stevens 1946) and the limitations on operations they each impose. It links the scales of measurement to well-known algebras, like monoid, group and field and motivates the introduction of the concept of homomorphism.

Chapter 4 FORMAL LANGUAGES AND THEORIES

Information systems use computers to manipulate symbols according to some formal rules, called programs. In this chapter we discuss the rules for the construction and manipulation of symbols, which are then used to represent facts describing the world.

Programs instruct computers to perform certain actions. Computer systems follow the rules laid down in the program when executing it. The result of a program with a given input are determined – if we see different results, then there must be additional inputs which vary between execution. Two computers may execute the same program differently: we say that they interpret the program differently; they use different dialects of the same programming languages, e.g., different extensions of the common standard language.

Programs are written in a formal language with a well-defined semantics. Programs have a different appearance and are more complicated than the axioms of the formal system we encountered in mathematics classes. Nevertheless, they are formal definitions of systems. In this book we concentrate on studying *formal systems*, which are introduced in this chapter.

Language = A set of symbols + rules for their combination.

Formula = A syntactically correct sequence of symbols in a language.

Theory = A formal language + rules concerning valid relationships within the language.

Formal System or Calculus = A language + rules for the transformation of formulae in other formulae.

1. FORMAL LANGUAGES

A formal language is a set of symbols that represents the vocabulary of the language and a set of rules how they can be combined to form legal well-formed formulae in the language. Formal languages are an abstract concept and the analogies to the vocabulary and the syntax of natural languages is limited. Natural languages have complex rules for the formation of words or sentences {de Saussure, 1995 #9510}. Applying the production methods described here for formal languages to natural languages has met with limited success(Chomsky 1980).

1.1 DEFINITION FORMAL LANGUAGES

A set of symbols (words, technically often called tokens) together with a set of rules for their combination, forms a *language*. The set of symbols is often called the alphabet and compares with the lexicon (vocabulary) of a natural language.

The rules for the combination can be called the *syntax* of the language—roughly equivalent to the grammar of a natural language. A symbol or valid combination of symbols constructed applying the production rules is a *well-formed formula*. When using a programming language, we speak of a syntactically correct program.

***Syntax**: rules of combination of symbols to well-formed formulae.*

In general, languages are thought of as producing linear sequences of symbols, similar to the text in a natural language. This is not a restriction; languages to construct spatial, two dimensional, arrangements have been explored in biology (Lindenmair grammars) and in spatial planning(Hillier and Hanson 1984).

1.2 STRINGS OF AN ALPHABET

A language is constructed from an alphabet, which is a finite set of symbols. These symbols can be combined to words; the set of all words of infinite length over an alphabet A is described as A*.

Alphabet A = {a,b}
A = {a, b, aa, ab, bb, ba, aaa, aab, aba, abb, baa, bab, bba, bbb, aaaa, aaab, ...}*

Strings are sequences of symbols over an alphabet. Strings with the operation *concatenation* ++, which merges two strings, form a *monoid*, which is a *semi-group* with a unit (namely the empty string ""). String concatenation is not commutative: $a ++ b \neq b ++ a$; ++ gives a non-commutative monoid; other monoids are commutative.

*Monoid <S, ++, ""> *	
associative	$a ++ (b ++ c) = (a ++ b) ++ c = a ++ b ++ c$
identity	$"" ++ a = a ++ "" = a$

The length of a string is the number of elements in it. The number of different strings in A*, where A contains *k* different elements with length exactly *l* is k^l. This can be seen by comparing the elements in A to the digits of the base k; with l digits we can form k^l different numbers.

String <S, length>	
distributive	*length (a) + length (b) = length (a ++ b)*

1.3 LANGUAGES DEFINED WITH PRODUCTION RULES

The alphabet for a language consists of three different sets of symbols:

- a close set of fixed symbols *T*, called the *terminal symbols,*
- a set of *non-terminal symbols N*, which are not part of the language (N and T must be disjoint),
- a special non-terminal symbol *S*, which is called the *start symbol.*

Only the terminal symbols appear in well-formed formulae of the language. For many languages, the terminal symbols are characters or numbers. Other languages have terminal symbols that are words, e.g., `BEGIN` and `END` in Pascal. The non-terminal symbols appear only in rules that lead to intermediate steps in the production of a language. For example, the language A* above is produced by

S ::= a | b | a S | b S.

Production rules explain how a symbol is replaced with other symbols in the course of the production of a well-formed formula of the language. Production rules have the form

n ::= u

where *n* stands for a non-terminal symbol and *u* is a sequence of terminal and non-terminal symbols. The production rules contain always a rule that translates the non-terminal start symbol *S* into a production. Production rules are applied repeatedly till all non-terminal symbols are replaced and only terminal symbols appear. An example for a simple language *RN* (which stands for a simplified form of *R*oman *N*umerals) with an alphabet containing the non-terminal symbols *S* and *N* and two terminal symbols *I* and + is given with two production rules:

In a production rule "|" stands for choice, either the left or the right part is selected.

Example language RN:
S :: = N | N "+" N *(1)*
N ::= "I" | "I" N. *(2)*

The rule (2) is recursive: the non-terminal N appears on the left and the right side. Therefore, the language can produce an infinite number of well-formed formulae, namely *I, II, III, ...* but also *I + II*, etc. Legal well-formed formulae in a language are all the sequences of terminal symbols that can be produced by repeated application of the production rules till the string does not contain any non-terminal symbols.

The production rules are usually written in Backus-Naur-Form (BNF). The BNF language is a formal language; it is a meta-language to describe other languages (the target language). BNF can be described in BNF, which is something like the famous Baron Münchhausen pulling himself out of a bog by his own hair! BNF uses the following terminal symbols:

```
::=      is replaced by, or, produces
|        or (select one or the other)
[]       optional (zero or one times)
{}       any number (zero, one, several times)
()       parentheses can be used for grouping
" "      quotes enclose terminal symbols
```

The production rules of BNF are:

```
syntax ::= { statement }
statement ::= identifier ::=" expression
expression ::= term { "|" factor }
term ::= factor { factor }
factor ::= identifier | "(" expression ")" | "["
expression "]" |
"{" expression "}"
identifier ::= string
string ::= character { character }
character ::= "A" | "B" | ... | "a" | "b" | ...
```

1.4 PARSING

Production rules are used to produce well-formed formulae, but are equally useful to determine if a given sequence of symbols represents a legal well-formed formula in the language. Compilers use production rules to analyze a given program and decide if it is a correct syntax, i.e. a well-formed formula in the programming language. An input text is *parsed* into tokens (see Figure 17 and Figure 18). In many cases, a program to parse the input can be produced automatically from the production rules. Parsing the string *III* of the language *RN* give the parse tree shown in (Figure 17), where the branches of the tree are labeled with the rule and the selection from the rule, which was used.

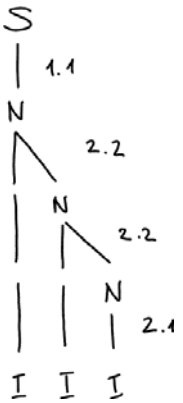


Figure 17: Parsing III from RN

1.5 EXAMPLE LANGUAGE: A SMALL SUBSET OF ENGLISH

A small language patterned after rules for the construction of simple English sentences should help to understand the concepts. The alphabet is:

Start Symbol: S
Non-Terminal symbols: {S, NP, VP, Det, N, V}
(standing for sentence, noun phrase, determiner, verb pharse, noun, verb respectively)
Terminal Symbols: {"the", "a", "Peter", "student", "professor", "saw", "met", "talked to"},

and the production rules are:

S ::= NP VP	*(1)*
NP ::= "Peter" | Det N	*(2)*
VP ::= V NP	*(3)*
Det ::= "the" | "a"	*(4)*
N ::= "student" | "professor"	*(5)*
V ::= "saw" | "met" | "talked to"	*(6)*

With this grammar, well-formed formulae can be derived using the steps:

1. Start with the start symbol

2. Repeat till no non-terminal symbols are left:
replace a non-terminal symbol with (one of the choices) of the right hand side of the associated production rule:

This produces for example:

```
S
NP VP                    rule 1
Peter VP                 rule 2.1
Peter V NP               rule 3
Peter saw NP             rule 6.1
Peter saw Det N          rule 2.2
Peter saw a student      rule 4.1 and 5.1
```

Other choices would give other well-formed formulae like:

```
A student talked to the professor
```

Parsing is the reverse process, where a well-formed formula of a language is given and the sequence of rules applied for its production are determined; one can think that the meaning of a well-formed formula is in the sequence of production rules used. Figure 18 shows the parse tree for "The professor saw Peter".

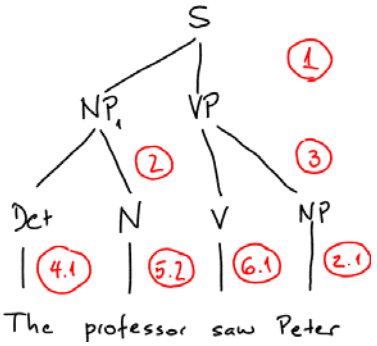


Figure 18: Parsing 'the professor saw peter'

1.6 THE LANGUAGE OF PROPOSITIONAL LOGIC

The language of propositional logic gives rules how well-formed logical formulae are constructed. Its terminal symbols are:

"(", ")", "not", "and", "or", "implies", "=", and symbols to represent propositions like P, Q, etc.
"and", "or", "implies", "=", and "not" are special symbols called Boolean operators

The non-terminal symbols are: *literal, wff (for well formed formula), variable, constant, operator, predicate, term*, and *atomic formula*, usually shortened to just *atom*. The language in BNF is:

wff ::= literal | (wff "or" wff) | (wff "and" wff)
| (wff "implies" wff) | (wff "=" wff)
literal ::= ["not"] atom
atom ::= proposition
proposition ::= "P" | "Q" | ...

The language just describes the appearance of a *wff* of propositional logic. Examples of *wff* are

not (P or Q) = not P and not Q
P or not P.

Examples for well-formed formulae are:

Mortal (Socrate),
Human (Socrate),
if Human (x) then Mortal (x).

1.7 LANGUAGE PRODUCES REPRESENTATIONS

The syntax of a language enumerates a set of words of the language. They are all distinct and can be used as constants to describe things. Such collections of representations are called *domains* and the symbols *tokens* or *(data) values*. Programming languages describe data types in a form similar to the BNF.

Consider the recursively defined data type *tree*; a *tree* is either a *leaf* or a *tree* combining two *trees*. This is defined in a program language (here Haskell (Peyton Jones, Hughes et al. 1999) as:

```
Tree = Leaf | (Tree, Tree).
```

The similarity to BNF is striking: Tree is a non-terminal symbol, *Leaf* and "(", "," and ")" are terminal symbols. A small tree would be *(Leaf1 (Leaf2, (Leaf3, Leaf4)))* (Figure 19).

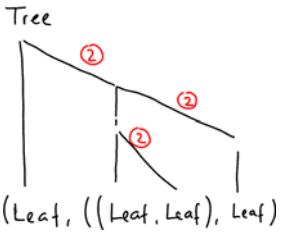


Figure 19: A simple tree

1.8 Information content of representation: the information measure of Shannon and Weaver

The information content in a well-formed formula of a language corresponds to the number of binary choices that are necessary to select this well-formed formula from all the possible well-formed formulae in the language of equal length. Assume a situation like Figure 20: A sender encodes a message (i.e. a well-formed formula) and the receiver tries to reconstruct the same message from the symbols received. The information content of the message is the minimal number of binary signals a sender must transmit to a receiver to enable him to recreate (re-select) a message from all possible messages (Shannon 1938; Shannon and Weaver 1949)

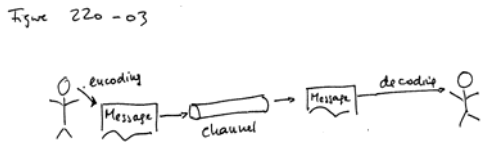


Figure 20: - Sender – channel – receiver

The information content of the representation is—following Shannon—

$$H = ld\ card\ (s)$$

where *ld* is the *logarithms dualis* and *card (s)* the number of different messages the sender may send and the receiver is prepared to receive. If a message has length *l* and is encoded with an alphabet of *k* symbols, then the number of different messages is k^l (see 1.2 above) and the information content is

$$H = ld\ k^l = l * ld\ k.$$

Information content increases linearly with the length of a message.

H is the maximum amount of information content a sequence of symbols (tokens) from one representation can carry. Information content is linear in the length of the string and the sum of the content of two messages is the same as the content of the concatenated message:

$$H\ (a) + H\ (b) = H\ (a\ ++\ b).$$

If the symbols are not selected with equal probability, then the information content of a representation must take into account the probability to select a symbol and the formula becomes:

$$H = -\ K \sum (p_i\ ld\ p_i).$$

This can be used to optimize the representation; tokens that are selected more often are represented by shorter and tokens that

are selected rarely are represented by longer strings of simpler tokens.

2. FORMAL SYSTEMS

A formal system or *calculus* consists of a language and rules for the transformation of representations into other representations. These transformations are called *evaluation*, if a complex expression is reduced to a simpler one. An example from arithmetic is: *3 + 5* is *8*; an example from logic is: "When it rains I use the umbrella" and "it rains" follows "I use the umbrella".

2.1 EVALUATION RULES FOR THE TRANSFORMATION OF REPRESENTATIONS

A formal system has rules for the transformation of well-formed formulae. These are rules, which say that two well-formed formulae are equivalent and we can transform one into the other(Carnap 1958). For example, logical proof uses the rule *modus ponens*, which says *"(A implies B) and A implies B"*. The language *RN* with the two following rules is a calculus. The well-formed formulae *II + III* can be evaluated to *IIIII*.

Evaluation is the simplification of an expression (wff) till it cannot be further simplified.

```
Rule 1: Ix + y = x + Iy
Rule 2: I + y = Iy.
II + III      apply rule 1 gives
I + IIII      apply rule 2 gives
IIIII.
```

Lower case letters stand for variables!

Rewriting is the principle behind the evaluation of functional programming languages (like Haskell(Peyton Jones, Hughes et al. 1999)) or logical expressions in the language Prolog(Clocksin and Mellish 1981).

2.2 PREDICATE CALCULUS

Predicate calculus models human rational thinking in a formal system(Lakoff and Johnson 1999; Lakoff and Núnez 2000). Logic discusses the deduction of the truth value of some combinations of logical propositions for which truth values are given(Sowa 1998p. 20). Only well-formed formulae have truth values and are either *true* or *false*, other combinations are just meaningless.

Examples:

& a *(not wff because & needs 2 parameters)*
(~b) & a *(wff)*

Propositional logic is a calculus—a symbolic computation based on fully defined rules. Predicates are expressions formed according to the rules for propositional logic (see above 1.6), like

q (x), p (a, b), which can be used to represent facts like *mortal (Socrates)* or relations like *son (Robert, Henri)*. The calculus of predicate follows rules that we intuitively accept as *logical.*

Syllogisms—formulae that are always true, independent of the values assigned to *P* and *Q* —are often used in explaining reasoning. Given the predicates *P*, *Q*, and *R* and the truth values *T* and *F* (for true and false, correspondingly) the following identities hold:

- Idempotent laws:
  P and P = P
  P or P = P
- Identity laws:
  P and F = F
  P or F = P
  P and T = P
  P or T = T
- Complement laws:
  not F = T
  not T = F
  P and not P = F
  P or not P = T
  not not P = P
- Commutative laws:
  P and Q = Q and P
  P or Q = Q or P
- Associative laws:
  P and (Q and R)=(P and Q) and R
  P or (Q or R) = (P or Q) or R
- Distributive laws:
  P and (Q or R =(P and Q) or (P and R)
  P or (Q and R)=(P or Q) and (P or R)
- Absorption laws:
  P and (P or Q) = P
  P or (P and Q) = P
- DeMorgan's Rules:
  not (P or Q) = not P and not Q
  not (P and Q) = not P or not Q
- Modus ponens:
  ((P implies Q) and P) implies Q
- Modus tollens:
  ((P implies Q) and not Q) implies not P
- Modus barbara:
  ((P implies Q) and (Q implies R))
  implies (P implies R)

Some useful terminology:
Given P implies *Q:*
converse*: Q implies P.*
inverse*: not P implies not Q,*
contrapositive*: not Q implies not P.*

a conjunction *consist of some propositions joined by AND,*

a disjunction *consist of some propositions joined by OR.*

In an implication, A => B, A is the antecedent*, B the* consequent.

These rules can be used to simplify complex expressions. For instance, the following Pascal conditional statement is difficult to decipher:

```
IF NOT ((name < > "Bob") OR (count < = 72)) THEN…
```

After the application of *DeMorgan's* rule, we obtain an equivalent expression that is much easier to read:

```
IF (name = "Bob") AND (count > 72) THEN…
```

Modus ponens is most often used for logical conclusions, such as

IF all humans are mortal
AND Socrates is human
THEN Socrates is mortal.

3. FORMAL THEORY

We are interested in a formal system where some facts and rules are interpreted as true and deduce other true statements. A formal theory is a mechanism whereby rules are employed to associate an initial set of well-formed formulae with all others. If

the appropriate associations can be made, the other *wff*'s are said to be *true* in, or *proven in*, or *deduced from*, the theory.

3.1 TRUTH VALUES

In a formal logic system, atomic formulae (*predicates*) are assigned values (called *truth values: True* or *False*). The initial assignment of truth values must be made by some agent external to the logic system; there is nothing intrinsically true about a formula. Mathematicians call an assignment of truth values an interpretation (in the sense of(Tarski 1977), similar as above in chapter 3). Most *wff* are either true (provable) or not; they can either be derived from the axiom set using the rules or they cannot. Gödel has shown that using unusual mechanism in an infinite universe, it is possible to construct formulae which neither can be proven, nor can we show their negation(Hofstadter 1985). Using a typed calculus avoids this problem.

3.2 BOOLEAN OPERATORS

The Boolean operators *and*, *or*, *not*, =, and *implies* are in the calculus defined by *truth tables*; these are equivalent to the syllogism given above (in 2.2); their meaning does not completely correspond to our everyday understanding of the corresponding natural language terms.

P	not P
true	*false*
false	*true*

The table above simply states that if *P* has a value *true* assigned, not *P* is false and vice versa. The next table shows the values obtained for *P and Q, P or Q, P = Q, P implies Q* and *Q if P*, for different assignments of *True* and *False* to *P* and *Q*.

P	Q	P and Q	P or Q	P = Q	P implies Q	Q if P
true	*true*	*true*	*true*	*true*	*true*	*true*
true	*false*	*false*	*true*	*false*	*false*	*false*
false	*true*	*false*	*true*	*false*	*true*	*true*
false	*false*	*false*	*false*	*true*	*true*	*true*

P implies Q is false if and only if *P* is true and *Q* is false. If *P* is true the results depend on the value of *Q* (which seems "logical" in the ordinary sense); however, if *P* is false, it doesn't matter what *Q* is! The result is always true, which may surprise and does not correspond to our natural language ideas of what implies means. What it is saying, however, is something like: "If you start with a false premise, anything is possible." This

demonstrates why consistency is important: a contradiction is always false and then anything can follow 'logically'. This is expressed as a syllogism that a contradiction is always false: *P and not P = F*.

3.3 AXIOMS AND THEOREMS

An *axiom* is a statement (a *wff*) in a theory that is assumed to be true. Any non-trivial theory must have one or more axioms. Sometimes the rules that explain how to prove some (non-axiom) *wff* are called the logical axioms of the theory. Usually, the axioms given above for first order predicate calculus are assumed. The other axioms are called non-logical. The non-logical axioms of the theory that do not contain variables are called ground axioms, ground rules, or simply *facts*.

A theory serves to test, whether a proposed statement, called a *theorem*, can be proven. If the proposed *wff* can be derived from the facts using the logical axioms it is then and only then a *true* statement (or logical model) in that theory.

Axiom = A fundamental statement in a theory—it needs no proof.
Logical Axiom = An association rule.
Non-logical Axiom = All other axioms.
Ground Axiom, Fact = A non-logical axiom that contains all constant values.
Theorem = A statement you wish to prove.

3.4 CLAUSAL FORMS

Since many *wff* can be logically equivalent, it is desirable to have a standard form. Every *wff* can be transformed to *clausal form,* which are implications where a number (possibly zero) of joint conditions implies a number (possibly zero) of alternative conclusions: The antecedent of the implication is a disjunction and the consequent is a conjunction:

A_1 *and* A_2 *and* A_3 *and … and* A_n *implies* B_1 *or* B_2 *or* B_3 *… or* B_m.

A_i and B_j are predicates, and $n, m \geq 0$. Since *A implies B* is equivalent to *B if A*, we write clauses in the following alternative clausal form:

B_1 *or* B_2 *or* B_3 *or … or* B_m *if* A_1 *and* A_2 *and* A_3 *… and* A_n

For example:

gfa (H, S) or gma (H, S) if pa (H, x) and pa (x, S).

Clauses are classified by the number of predicate terms in their consequent as:

- definite (if there are zero or one term) or $m <= 1$
- indefinite (if there are two or more terms). $m > 1$

The definite clauses are called *Horn clauses*. In the case where $m = 1$, and $n = 0$, we have a definite clause that represents a *fact* or *ground axiom*.

fa (A, S) if ()

Since no antecedent is required for the consequent, it is always true. Usually the empty *if* is discarded in this situation. Definite clauses where $m = 1$ and $n > 0$ are called *rule clauses*. They represent a logical axiom.

B_1 *if* A_1 *and* A_2 *and* A_3 … *and* A_n

For example:

gfa (x, z) if fa (x, y) and fa (y, z)

With a definite clause that has no consequent (i.e., $m = 0$, $n > 0$), the antecedents are considered to be *negative facts*, that is, facts that are known to be false.

if fa (A, I)

The empty clause has m, $n = 0$; it is always false by definition.

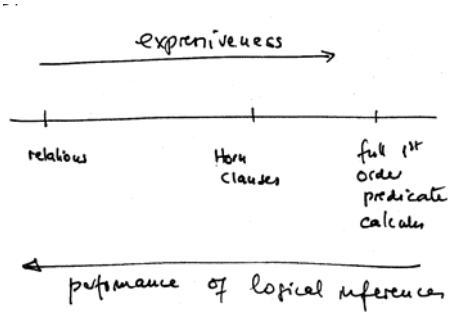


Figure 21: Trade-off between expressiveness and performance

The next sub-section shows two mechanical (programmable) algorithms to produce a logical proof. They expect the input as Horn clauses. Horn clauses seem to be a nice compromise between expressability and performance (Figure 21); more expressive languages lead to more complex and slower processing. Horn clauses are sufficient to express definite facts, but it is not possible to include negative statements. Relations are even less expressive; they allow only collections of facts, but deductions are much faster and reduce to search in the facts (see part 5).

3.5 PROOFS

Given a set of formulae, which are assumed as *true*, a proof is a sequence of logical transformation using the deduction rules given above (section 2.2), which show how the hypothesis can be derived from the axioms. The given true formulae are called *axioms*; the formula to derive is the *hypothesis*.

3.5.1 Unification

For each step in the process, unification between the variables and constant in the formula to proof with the axiom that is used in this step is required. Variables can be matched with variables, and variables can take on the values of a constant expression, but it is not possible to unify a constant with another constant or a variable already unified (bound) to a constant with another constant.

3.5.2 Example theory: family relations

The theory we build is representing some facts about a family written as Horn clauses. Constants will be marked by upper case

Hint for interpretation:
fa = father
gfa = grandfather

symbols (*A, B, C, ...*), variables with lower case letters (*x, y, z...*).

fa (A, S)
fa (H, A)
fa (G, H).

From these facts and the rule

gfa (x, z) if fa (x, y) and fa (y, z),

we can conclude using formal symbol manipulation without reference to any interpretation of the symbols involved that the following statements are true:

gfa (H, S)
gfa (G, A).

The next two subsections show how to proof the first of these formulae.

3.5.3 Forward chaining—from facts to conclusions

Forward chaining uses *modus ponens*:

fa (x, y) and fa (y, z) implies gfa (x, z)

(1) Select the first fact and substitute it into the logical axiom (x=A, y=S), gives

fa (A,S) and fa (S,z) implies gfa (A,z)

(2) Find an axiom that can be unified with the next predicate in the antecedent. There is no fact that can be unified with *fa (S, z)*, we have to *backtrack* and return to (1). Select another fact: *fa (H, A)* gives substitutions *(x=H, y=A):*

fa (H,A) and fa (A,z) implies gfa (H,z).

Now, the fact *fa (A,S)* can unify with the predicate *fa(A,z)* and gives substitution (*z=S*):

fa (H,A) and fa (A,S) implies gfa (H,S) q.e.d

Reasoning with modus ponens starts with the facts and combines these in all possible ways till it reaches the theorem for which we search a proof. This works in small examples, but the number of possible combinations to explore grows exponentially in practical applications (combinatorial explosion). The algorithm has no guideline in which direction to go for interesting combinations and explores mostly ‘blind allies’.

3.5.4 Backwards chaining—from conclusions to supporting facts

A more effective form of reasoning is ‘backward chaining’, that is, starting with the question *gfa (H, S)*—which is only one—and tries to find facts to prove it. In this case, we use *modus tollens*

((P implies Q) and not Q) implies not P

and try to proof the negation of the question. Given the question which value of *u* stands in relation *gfa* to *H*

gfa (H, u).

We start with its negation (i.e. stating that nothing stands in relation *gfa* to *H*):

not gfa (H, u)

using the (only possible) rule

gfa (x, z) if fa (x, y), fa (y, z)

and have substituted *x = H*, *z = u:*

gfa (H, u) if fa (H, y), fa (y, u).

We search now for a fact that unifies with *fa (H, y)*, which we find only with the substitution *y = A*, this leaves us with

gfa (H, u) if fa (H, A), fa (A, u).

Now we search for a fact that unifies with *fa (A, u),* which we find only with the substitution *u = S*

gfa (H, S) if fa (H, A), fa (A,S).

not gfa (H, u) is not true, because *gfa (H, S)* is provable. This leaves us with the useful result that *H* is in the relation *gfa* to *S*. As you have noticed, the search for 'useful' facts is automatic; if none would have been found, we had concluded that nothing stands in the relation *gfa* to *H*. Languages like Prolog (Clocksin and Mellish 1981) using such backward chaining.

3.5.5 Comparison

Forward chaining uses *modus ponens* and moves from facts to conclusions, backward chaining uses *modus tollens* and moves from conclusions to facts. Both are used in AI and are applicable to geographic expert systems(Frank, Robinson et al. 1986; Frank, Robinson et al. 1986; Frank, Robinson et al. 1986; Frank, Hudson et al. 1987; Frank and Robinson 1987). Forward chaining gives to a set of facts all possible conclusions; it works only for small numbers of facts, because the number of possible conclusions increases with the number of facts exponentially. Backward chaining searches for the facts that support a given conclusion. It is useful when a conclusion is given and we need to test, if it is following from a collection of facts and rules; backward chaining is selective and can be used even with large collections of facts.

3.6 LOGIC WITH MORE THAN 2 TRUTH VALUES

Usually the range of the values is restricted to either *True* or *False*, although multi-valued logics have frequently been employed, e.g., with values: *True*, *False*, *Maybe*; or real number values ranging between *0* and *1* representing various degrees of probability of a statement, or certainty about classification, so-called Fuzzy Logic(Zadeh 1974).

For example, the three valued logic of Lukasiewicz has 3 truth values: True, Neutral, and False. Neutral can be loosely interpreted as 'it is possible that'. The truth tables are:

P	not P
true	*false*
neutral	*neutral*
false	*true*

P	Q	P and Q	P or Q	P implies Q
true	*true*	*true*	*true*	*true*
true	*neutral*	*neutral*	*true*	*neutral*
true	*false*	*false*	*true*	*false*
neutral	*true*	*neutral*	*true*	*true*
neutral	*neutral*	*neutral*	*neutral*	*true*
neutral	*false*	*false*	*neutral*	*neutral*
false	*true*	*false*	*true*	*true*
false	*neutral*	*false*	*neutral*	*true*
false	*false*	*false*	*false*	*true*

3.7 TEMPORAL LOGIC

Logic described so far treats predicates that are not changing. To reason about changes requires either a temporal logic, of which several are known (Sernadas 1980; van Benthem 1983; Everling 1987; Galton 1987) and which are difficult to use. Temporal logic includes four predicates (as defined by Prior [ref missing]):

P – it has at some time been the case that …
F – it will be at some time be the case that …
H – it has always been the case that …
G – it will always be the case that ..

of which *P* and *F* can be defined in terms of *G* and *H*:

P a = not H (not a)
F a = not G (not a).

Alternatively, situation calculus separates a changing world in ‘snapshots’, called situations and then describes each of them separately, assuming that the constant symbols stands for the same individuals at different times(McCarthy 1996). An improved version of situation calculus was presented by Reiter(Reiter in preparation); however, it uses some ‘extralogical’ devices to arrive at a usable structure. Bittner has compared situation calculus as method to describe a GIS problem (real estate cadastre) (Bittner and Frank 1997)with an algebraic description and found that they are different formal expressions for essentially the same constructions.

3.8 VARIABLES AND QUANTIFICATION

Logical formulae are written using variables and it is usually implied that the rule should be valid for all values of these variables. This is expressed with the *all quantor*:

$$\forall x : human(x) \rightarrow mortal(x)$$

The *existential quantor*, states that there is at least one *x* such that the formula is true:

$$\exists x : human(x)$$

A variable occurring in a quantor is said to be bound. It is customary to drop the all quantors whenever obvious and write only the existence quantors.

4. ORDER OF A LANGUAGES

Languages can be classified by orders. We pay attention what role variable symbols—which are bound by quantifiers—can play:

- A zero order language has no variables, only constants.
- A first order language has variables, which stand for objects, but not for predicates or functions.
- A second order language has variables that can stand for objects, predicates, or functions (sometimes called higher order).

Classical logic, as used by philosophers, is (mostly) first order. Functional programming languages are an example of second order languages(Backus 1978; Bird and Wadler 1988). In principle, all formulae can be expressed in first order languages(McCarthy 1985), but the expressions become complicated and difficult to understand.

5. TYPED LANGUAGES

It is useful to subdivide the constants in disjoint sets. One then says that the constant *x* has type *t*, for example, the constants *Andrew, Stella*, etc. all have type *Human*. The predicate *father* establishes a relation between two constants of type *Human*, and is meaningless if connected with constants of other types. Variables in formulae have corresponding types and formulae can be checked if they are consistently typed.

The type information that belongs to some formulae is called its signature; we write it after a double colon: *fa :: Human -> Boolean*.

A typed language is not more expressive than an untyped one, but typed formulae exclude many non-sense formulae from consideration. Typing is not necessary when considering simple examples with few formulae, but becomes useful when describing large systems. Most modern programming languages are typed. A compiler for a typed language can formally check not only the syntax of the program, but also assure that the program is consistently typed. This excludes a large class of errors from occurring when the program is executed(Cardelli 1997).

6. CONCLUSION

Languages produce the representations which stand for the facts of the world. Considering the size of the language produced by a set of rules and comparing it with the variation in the phenomenon we need to represent is often useful: we may find that a language is too small to represent all the differences we are interested in or it may be too large and contain multiple representations for the same facts or many symbols which are not representing anything.

Logic transforms expressions we consider true into other true expressions; it mimics the human rational thinking. Logical deduction can be automated, but the most powerful reasoning engines are slow and the fast one are restricted – sometimes only simple facts and no rules can be dealt with, sometimes negation is not included in the language. For practical applications a trade-off between speed and expressive power must be stricken.

REVIEW QUESTIONS AND EXERCISES

- What is a production rule? Give an example.
- What are the elements of BNF? What is it used for? Give examples and explain them.
- What is the difference between formal language and formal system?
- What is the relationship between language and representation?
- Explain the information measure of Shannon and Weaver.
- Why are the following well-formed formulae of the language 'small subset of English' not well-formed:

  Simon saw Peter.
  A Peter saw student.

- What is a parser? What does it produce? How does it use production rules?
- Extend the language "Small Subset of English" to include ‘and’ in well-formed formulae like "Peter and the professor talked to a student". Give the parse tree for the well-formed formula.
- Express the conditions for a construction site as conditions on size, reachability, exposition, etc. and simplify the expression using the formulae for Boolean expressions.
- Simplify the expression for leap years:

```
leapYear y = ((mod y 4 == 0) && (mod y 100 ≠ 0)) ||
     (mod y 1000 == 0)
```

- What is *modus ponens*? Give an example.
- What is the difference between forward and backward chaining? Give an example for each.
- Why is the *III* + *I* = *II* wrong (using the rules stated above)?
- What is meant with quantification of a logical formula? Give an example.
- Extend the ‘family example’ with the fact *fa(R,H)*. Demonstrate the deduction of *gfa(R,A)* using backward chaining.
- What is a Horn Clause?
- What are truth tables?
- What is the difference between a typed and non-typed language? Is there a difference in expressiveness?
- Show that *if A then B* and *B implies A* is equivalent using the respective truth tables.

Chapter 5 ALGEBRAS AND CATEGORIES

Logic describes properties of things; algebras focus on the notion of transformations (mappings) from states to states. This seems an attractive mathematical tool for geography that studies processes in space and time(Abler, Adams et al. 1971).

Abstract algebras give a definition to the previously introduced notion of *structure*, which was mentioned in chapter 3. Category theory generalizes the notion of algebra. It is well-known, that everything can be expressed in logic(Lifschitz 1990), but also in algebra. The purpose here is not theoretical but eminently practical: to find a mathematical tool that leads to a description of a complex system like a GIS that is compact and easy to understand. The following chapter 6 demonstrates its use to describe the measurement scales of Stevens(Stevens 1946).

1. INTRODUCTION

Algebra, by which I always understand abstract or universal algebra(Whitehead 1898), is a development of the 20th century. It has emerged from a view that algebra deals with the properties of numerical operations to investigations of the structure of operations. Algebra does not deal "primarily with the manipulation of sums and products of numbers (such as rationals, reals, or complex), but with sums and products of elements of any sort—under the assumption that the sum and product for the elements considered satisfy the appropriate basic laws or 'axioms'" (MacLane and Birkhoff 1967, vii).

"an algebraic system ... is thus a set of elements of any sort on which functions such as addition and multiplication operate, provided only those operations satisfy certain basic rules" (Mac Lane and Birkhoff 1991, p.1).

The development in mathematics in the 20th century has stressed generality. Operations, which do not necessarily satisfy the laws of sum and product are investigated. Increasingly, separate parts of mathematics are dealt with in an algebraic fashion; we will introduce Boolean Algebra, in contradistinction to the closely related Boolean operators of propositional logic or predicate calculus shown in the previous chapter 4. Later we use algebraic topology (Part 8). Logic is closely related to the theory of databases(Gallaire, Minker et al. 1984); with equal justification one can say that algebra and category theory is the theory of computation(Asperti and Longo 1991).

Software engineering has appropriated category theory to construct tools to write specifications(Guttag, Horning et al. 1985). Operations are divided into *constructors*, which produce all the different elements in the domain and in *observers*, which report the differences between the expressions(Parnas 1972). The meaning of the operations is defined as properties of the result, as these are observed with other operations in the same algebra; to define the intended behavior of operations, we give expressions of the form

obs1 (constr1 ...) = value,

where *obs1* and *const1* are operations of the algebra(Ehrich, Gogolla et al. 1989; Loeckx, Ehrich et al. 1996). This allows definitions independent of other previous definitions and circumvents the grounding problem of logic definitions by enumeration of properties.

Algebraic structure captures the essence of the semantics of operations and objects.

Algebra discusses the *structure* of operation and defines precisely what is meant by structure. Structure of operations means properties of operations that are independent of the objects the operations are applied to. Algebra describes the 'structure' of a real world system in a precise way and independent of the representation. It is possible to describe the structure of complex real world systems—e.g., a coke vending machine—as an algebraic system and investigate its properties. The descriptions of the structure are independent of the realizations that behave the same; we say the descriptions are determined up to an isomorphism.

2. DEFINITION OF ALGEBRA

Algebra consist of
- a set of domains (sets of elements),
- a set of operation names and signatures,
- a set of axioms that describe the properties of the operations.

An algebra describes a class of *objects and their behavior*, and is closely related to the object-oriented discussion of software engineering(Guttag and Horning 1978). An abstract algebra consists of a collection of domains, and operations with axioms, describing their properties. One could differentiate between the theory described by an algebra, the algebra, an abstract data type and models for the algebra; but this seems not necessary for present purposes and following Erich et al. (Ehrich, Gogolla et al. 1989)and (Asperti and Longo 1991) I use the term algebra broadly.

The next subsection gives as examples the basic algebraic structures that will be used later. We have already seen monoid (chapter 4). Here we introduce:

- Group
- Natural Numbers (integers)
- Boolean Algebra
- Sets
- Category

The next chapter will give:

- Equality, Order
- Ring and Field

Later chapters will use

- Lattice

We will always assume that an equality relation is defined for the domains and that all variables in axioms are implicitly all-quantified; existential quantification, if necessary, is stated.

2.1 GROUP

A group is an algebra that has an operation (written here as +), an inverse for this operation (written as -) and a unit value (written here as *0*). The standard example is integers with *plus, minus,* and *zero*, but group is also an important algebraic structure in geometry. For example, translation (or rotations) in geometry form a group; the zero element is 'translation by the zero vector' (i.e., not doing anything).

Group <+,-,0>	
associative	$(a+b)+c = a+(b+c)$
unit	$0+a = a+0 = a$
inverse (- a):	$a + (-a) = (-a) + a = e$

Many important groups are commutative ($a + b = b + a$) and are called commutative or Abelian group—honoring the mathematician Niels Abel (1802 - 1829). Ordinary addition is commutative and integers with plus form an Abelian group.

2.2 NATURAL NUMBERS

The natural numbers are as fundamental as points and lines in geometry. The axioms for geometry were studied by the Greek and formulated by Euclid around 300 BC in his *Elements(Heath 1981; Adam 1982; Blumenthal 1986)*. In contrast, an axiomatic definition for natural numbers was only given in the later 19th century by Peano(Kennedy 1980).

The properties of natural numbers with basic arithmetic operations (+, -, *, /) are described as an integral domain (see next chapter). We can construct a model for natural numbers

with a representation using the simple language *RN* (from chapter 4). The axioms following the original description by Peano include a definition for equality of two numbers and for addition of two numbers:

1. 1 elem N
2. for all m (m elem N) exist a unique m' (m' elem N), called the successor of m.
3. for each m elem N, m'≠ 1 (That is, 1 is not the successor of any natural number).
4. If m, n elem N such that m' = n' then m=n
5. let K be a set of elements of N. Then K=N provided the following conditions:
(i) 1 elem K
(ii) if k elem K, then k' elem K.
Def. Addition: Let m, k be arbitrary elements of N. We define m + 1 = m'. If m + k is defined then m + k' = (m+k)'
(McCoy and Berger 1977).

2.3 BOOLEAN ALGEBRA

A Boolean algebra with just two constants (customary notations are "*T*" and "*F*" or *True* and *False* or *0* and *1*) and a unary operation *not* and binary operations *and*, *or*, *implies*, gives equivalent rules to what was before described (chapter 4) as Boolean Logic (named after George Boole 1815-1864). Laws like *de Morgans* law *(not (a or b) = (not a) and (not b))* and similar can be added here as well.

Boolean Algebra <{T, F}, and, or, not>	
	not :: b -> b
	and, or, implies:: b -> b –> b
self-inverse	*not (not p) = p*
associative	*a and (b and c) = (a and b) and c = a and b and c*
	a or (b or c) = (a or b) or c = a or b or c
commutative	*a and b = b and a*
	a or b = b or a
units	*a and T = a*
	a or F = a
inverse	*a and (not a) = F*
	a or (not a) = T
distributive	*a and (b or c) = (a and b) or (a and c)*
	a or (b and c) = (a or b) and (a or c)

The logical operations can be defined as numeric functions as follows: represent False by 1 and True by 2, then

and (a,b) = min (a,b),
or (a,b) = max (a,b),
not a = 3-a;

this approach is useful when allowing more than 2 truth values in a logic system(Sinowjew 1968).

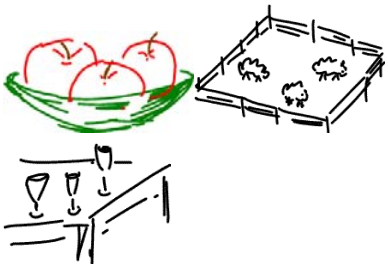


Figure 22: Examples of real world sets

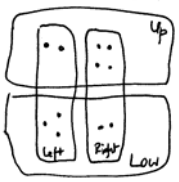


Figure 23: Venn diagrams of four sets intersecting

Figure 24: Sheep grazing on a hill

2.4 SET

Sets are an abstraction from the collection of elements as we encounter them in real life everywhere—fruit in a bowl, sheets of paper in a folder, glasses on a table, etc. (Figure 22). A simplistic set theory would construct only sets of real world objects, such that each object can only be in one set at a time. Then we have

card (a) + card (b) = card (a ∪ b) *if a intersectoin b = empty.*

.

This is too restrictive: an element can be in more than one set at a time, but it cannot be multiple times in the same set (a structure that permits multiple memberships is called a bag or multiset). Venn Diagrams are a useful tool to visualize sets and operations with sets. For example the intersection of the sets 'left paddock', 'right paddock', 'down', 'up' from Figure 24 is shown Figure 23.

Sets give another example of a Boolean algebra. If the rules for sets are restricted to just two values all-set (for *True*) and null-set (for *False*), then *union* corresponds to the Boolean operation *or* and *intersection* corresponds to Boolean *and*.

Set < ∪, ∩, complement, ∅, all>

associative	$a \cup (b \cup c) = (a \cup b) \cup c$
	$a \cap (b \cap c) = (a \cap b) \cap c$
commutative	$a \cup b = b \cup a$
	$a \cap b = b \cap a$
identity	$a \cup \varnothing = a$
	$a \cap all = a$
inverse	$a \cup (comp\ a) = all$
	$a \cap (comp\ a) = \varnothing$
distributive	$a \cup (b \cap c) = (a \cup b) \cap (a \cup c)$
	$a \cap (b \cup c) = (a \cap b) \cup (a \cap c)$
	involution $comp\ (comp\ a) = a$
idempotent	$a \cup a = a$
	$a \cap a = a$
$\varnothing$	$a \cup all = all$
	$a \cap \varnothing = \varnothing$
absorption	$a \cup (a \cap b) = a$
	$a \cap (a \cup b) = a$

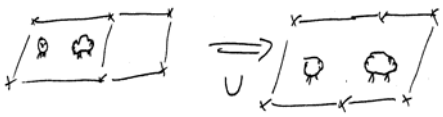


Figure 25: Union with the empty set

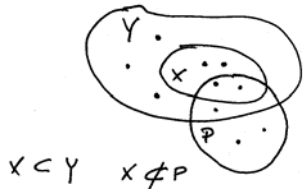


Figure 26: Subset relations form a partial

Figure 27: Set containing sets

The operation cardinality computes the number of elements in a set; the following rules apply:

card (a ∪ b) = card a + card b –card (a ∩ b)
0 < card (a ∩ b) < min (card a, card b)

It is possible to determine if an element *a* is in set *A* with a membership function or *element of* (written as ∈) relation. The expression *'a ∈ A'* is true if *a* is an element of set *A*.

A set *X* is a *subset* of another set *Y* if every element of *X* is also an element of set *Y*, written as $X \subset Y$. Venn diagrams express subsets relations by inclusion (see Figure 23 above). The converse relation is called superset ($Y \supset X$). Subsets form a partial order (Figure 26).

Mathematicians have constructed sets that contain sets and are different from sets just containing the elements in them; the set on the left of Figure 27 contains three elements, namely the sets *E*, *F*, and *G*, whereas the set on the right contains the 12 elements which were in *E*, *F*, and *G*. Allowing unrestricted sets and set membership can lead to antinomies; for example, does the set, that is defined as containing all sets that do not contain itself, contain itself? Operating in a typed universe, these problems cannot occur, because as *set of x* is a different type than the *set of (sets of x)*.

3. DUALITY

The axioms for Boolean algebra and for sets exposed a regularity: every axiom formulated for the operation *union* had a corresponding axiom for *intersection* (respective *and* and *or*). A valid formula can be converted into another valid formula, if we systematically exchange every operation for the dual operation and equally exchange the units: the all-set (True) becomes the empty set (False) and vice versa. Duality could have been used to reduce the number of axioms stated; it will become more useful later replacing operations difficult to compute with others that are easy (see chapters 19 and 30).

Duality for set:
∪ <-> ∩
allset <-> null set

4. FUNCTIONS ARE MAPPINGS FROM DOMAIN TO CODOMAIN

A function from a domain *A* into a codomain *B* maps values from *A* to values of *B(Gill 1976)*. A function assigns to a single value from *A* only a single value from *B* (unlike relations, which can have multiple result values—see later chapter 16). Computer science speaks of the domain and codomain as types(Cardelli

1997); the signature of a function *f : A -> B* gives the domain and codomain. The application for single element *x* is written as *x |-> f (x),* and *f (x)* is the element assigned to *x*.

Terminology:
A function f : A -> B
maps from domain A to codomain B.

Functions with more than one input or more than one result can be seen as function of a single input and single result, if the inputs or results are considered as tuples. The function *a + b* can be transformed into the function plus *(a,b)*, which has a single input, namely the pair *(a,b)*.

4.1 TOTAL FUNCTIONS

A function *f : A -> B* that has a result for any value in its domain is called total(Figure 28). Programmers prefer total functions because they do not require a test that a function value can be obtained. Non-total functions produce results only for a subset of the values in the domain (**Error! Reference source not found.**).

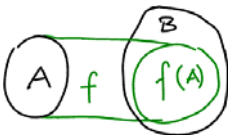


Figure 28: Total Function with image f(A)

*Functions are **total** if they take every element of S to an element of T; they are not total if there are elements of S that are not mapped.*

Examples: Increment is a total function that adds *1* to any number. Division is partial, as division by *0* is not defined (Ehrich, Gogolla et al. 1989; Mac Lane and Birkhoff 1991, 6).

4.2 INJECTIVE FUNCTIONS (INTO)

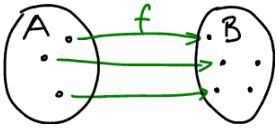


Figure 29: Injective Function (inverse is partial)

If a function has an inverse function, then it is an injection: for each value of the domain there is a different value in the range (Figure 29): *a ≠b implies f(a) ≠f(b)*. Injective functions carry distinct elements in the domain to distinct elements in the codomain(Gill 1976, 53). Figure 30 gives an example of a function which is not injective. An injective function has an inverse function from the codomain to the domain.

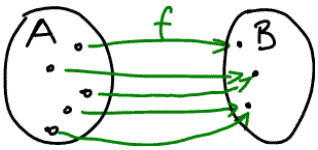


Figure 30: an example of a function which is not injective

4.3 SURJECTIVE FUNCTIONS (ONTO)

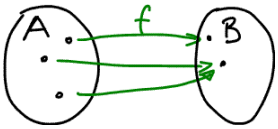


Figure 31: Surjective Function

A function f : A -> B is *surjective* if every element of *B* is the image of some element of *A* (Figure 31). The image is the whole codomain. Figure 32 gives an example of a function which is not surjective.

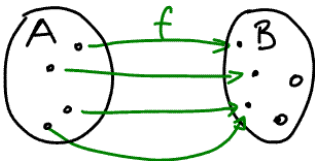


Figure 32: example of a function which is not surjective

4.4 BIJECTIVE FUNCTIONS (ONE-TO-ONE)

If a function is surjective and injective, it is called *bijective* (Figure 33). Such functions have inverses that are total. We will later see that this classification is useful not only for functions, but generalizes to relations (see chapter 16).

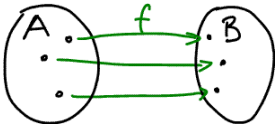


Figure 33: Bijective function (has inverse, which is total)

A function *f: S -> T* is **injective** if no two inputs are mapped to the same result.
A function is **surjective** if every element of its codomain T is the result of some element in the domain S.
A function is **bijection** if it is both injective and surjective

5. ALGEBRAIC STRUCTURE

Algebras describe structure independently of an implementation or previous understanding. The same algebra describes the behavior of a class of different things if their behavior is structurally equivalent. For example, the operations with counts and operations that apply to the result of the counting, are the same, independent of what we count: beers, sheep, matches, gold bars, whatever.

The structure is described in form of axioms, which can often be expressed as the observation of the result of on operation in terms of other observations. For example all numbers useful for counting must have the structure of a *group* with the rule that adding zero to a number yields the same number: *a + 0 = a.* Numbers represented as Roman numerals, Arabic numbers, or as binary numbers in a computer work the same. The algebra describes behavior '*up to an isomorphism'*, meaning it describes many things that behave, under the limited perspective of the operations defined in the algebra, the same. Differences that can not be observed with the operations defined in the algebra are not relevant.

5.1 EXAMPLE— COUNTING

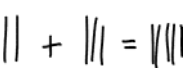


Figure 34: 2 + 3

The definition of equality or addition on natural numbers as defined by Peano (subsection 2.2 above) is only of interest as it is useful for solving real world problems. How many beers do I have to pay if my count reads *II1* and my friends *II*? Using the rules from *RN* (see chapter 4) *II1 + II = IIII1*, which is *5* in ordinary language (Figure 34). The algebraic definition of addition corresponds to the natural adding of counts. This correspondence was introduced before when we discussed information systems as a morphism (see chapter 3).

The homomorphism h : A -> B carries also the operations f : A -> A to operations f' : B -> B, such that h (f (a)) = f' (h (a)).

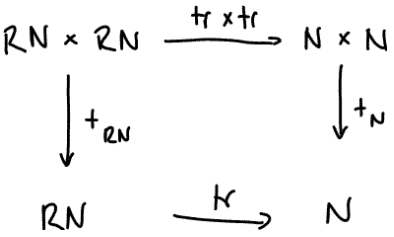


Figure 35: Commutative diagram

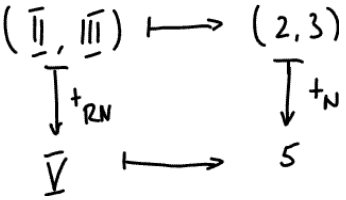


Figure 36: Two alternative paths to compute II + III

An algebra with axioms describes a structure independent of the carriers or the names of the operations

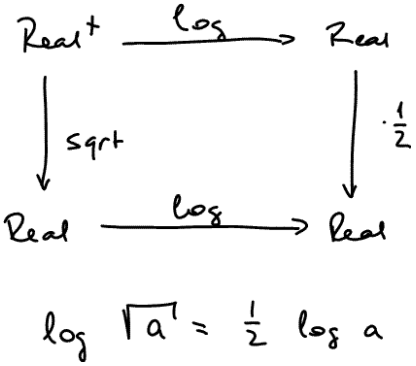


Figure 37:The law of exponents defines a morphism

5.2 HOMOMORPHISM BETWEEN ALGEBRAS

A morphism, is a mapping between things of two types which preserves its algebraic structure. The addition (operation + above) is the same if applied to Roman numerals *II + III = V* or to Arabic numbers *2 + 3 = 5* (Figure 36), but it is also the same if we take any set with cardinality *2* (e.g., a pair of sheep, Figure 34) and merge it with a set with cardinality *3* (e.g., another flock of 3 sheep).

The algebraic structure of addition is preserved across the mapping *F* from one kind of numbers to the other: We can add the Roman numerals and transform (map) the result to Arabic numbers or we can transform first to Arabic and then do the addition, the result is the same (Figure 35). Category theory studies a generalized concepts of morphism and shows morphism succinctly as a commutative diagram (Figure 35) (Barr and Wells 1990; Asperti and Longo 1991; Mac Lane and Birkhoff 1991).

A homomorphism does not imply that the operation maps to the 'same' operation. The example with different counts (Figure 34) may be misleading: the operation was two different kinds of *add*, once with *RN*, once with *N*. Logarithm gives a different, familiar example (Figure 37).

5.3 DEFINITION MORPHISM AND COMMUTATIVE DIAGRAM

Mac Lane and Birkhoff explain a morphism for a binary operation as: “Let * be a binary operation on a set X, while ▫ is another such operation on a set *X’*. A morphism *f: (X,*) -> (X’, ▫)* is defined to be a function on *X to X’* which “carries” the operation * *on X* to ▫ *on X’*, in the sense that

$$f(x * y) = (f x) \square (f y)$$

for all $x, y \in X$. On the left (Figure 38), one applies to an element $(x,y) \in X \times X$ first the operation *, then the function *f*; on the right one applies first $f \times f$ (i.e., apply *f* to both elements in the pair) and then ▫. In other words, *f* is a morphism if and only if the diagram below is commutative." (Mac Lane and Birkhoff 1991p. 37). This can be generalized to unary functions as:

"If (X,h) and (X', h') are sets with unary operations h: X -> X, h': X' -> X', a morphism f: (X,h) -> (X', h') of unary operations is a function f: X -> X' with f (h (x)) = h' (f (x)) for all x ∈ X. " (Mac Lane and Birkhoff 1991 p. 38) and to functions with more than two arguments.

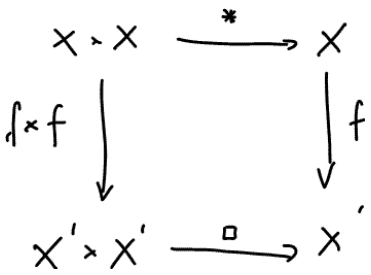


Figure 38: Commutative diagrams (after(Mac Lane and Birkhoff 1991, 37))

A categorical diagram is said to be commutative if we can travel both paths and arrive at the same result, often from top left to bottom right. In Figure 37 it does not matter if we take the logarithm first and divide by two (right and down path), or if we take the square root first and then compute the logarithm (down and right path).

Morphisms are classified by considering the type of the function f. A morphism *f: (X, h) -> (X', h')* is a

monomorphism	if the function f is injective:
epimorphism	if the function f is surjective;
isomorphism	if the function is bijective.

Isomorphisms are very important in computer programming as they allow a lossless transformation forwards and backwards between two representations and the corresponding operations. For example, computers are faster adding binary representations of integers than integers represented as Roman numerals or strings of digits. It is customary to represent integer numbers in most cases as binary numbers and have all operations executed with them. Computer representations are not isomorphic to integers, because only numbers up to a certain magnitude can be represented; the transformation is isomorphic only for the restricted domain, but this is nearly always sufficient.

injection ->	*monomorphism*
surjection ->	*epimorphism*
bijection ->	*isomorphism*

5.4 APPLICATION TO INFORMATION SYSTEMS

The representation relation between the things in the world and the things in an information system introduced in chapter 3 is not a simple static mapping, relating objects in the world to objects in the program; in Figure 34 the two sheep map to the numeral *II*. We must also map the operations in the world—*merging* the two flocks of sheep—to the operations with numbers in the computer—the *addition*. We need two kinds of mappings: objects to representation and operations we can perform in the world mapped to operations applied in the information system to the representations. This is exactly what a morphism does. To be useful, the outcome of an operation in the world and the corresponding operation in a computer must correspond. I called this—in analogy to the commutative diagrams of category theory—closed loop semantics (Figure 39) : applying the action

to the world or to the mental image obtained by observation must result in the corresponding result(Frank 2001; Frank 2003).

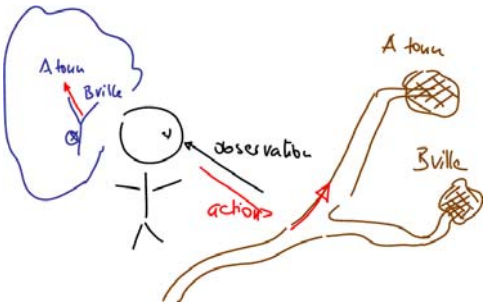


Figure 39: Closed loop semantics: the observations are connected to actions

5.5 MODELS

Algebras are abstract constructions. If we want to implement and experiment with an algebra, we have to build a model with carriers we can represent and operate on. A representation for an algebra is a *model* of this algebra. Roman numerals are a model of natural numbers. For computational models, the representations are computer data types (see chapter 4). Models of algebras with different representations have the same behavior, because the behavior is the abstract property of the algebra. Some models of algebras are special (initial algebra, terminal algebra, Herbrand model) but this is not of importance in this context; we will tacitly assume initial algebras as models for our specifications(Ehrich 1981; Loeckx, Ehrich et al. 1996).

5.6 MORPHISMS AS A METHOD TO CONNECT ALGEBRAS

We have encountered two morphisms in the previous chapter, namely the operation to determine the length of a string *length* and the operation to determine the information content of a string *H*. For both we have stated that they combine with string concatenation ++:

length (a) + length (b) = length (a ++b)
H (a) + H (b) = H (a ++ b)

These are both morphisms that map strings to numbers and concatenation to addition, such that the two diagrams commute (Figure 40, Figure 41).

Morphisms connect algebras.

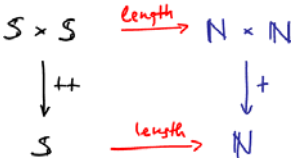


Figure 40: Length is a string morphism

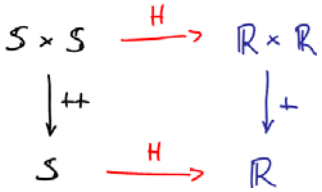


Figure 41: Information content H is a string morphism

6. IMAGE AND KERNEL

The *image* of *G* in *H* is the set of values that are the result of applying *Φ* to all values in G. The image of *Φ* has the same structure as *G* (for example if *G* is a group then the image of *Φ* is a subgroup of *H*).

If the algebraic structure has units, then we may ask, what are all the elements of *G* that map to a unit of *H*? This set is called the *kernel* or *null space* (i.e. the values in *G* which solve the equation *Φ (g) = 0*). It indicates how much this morphism "collapses" *G* (Mac Lane and Birkhoff 1991, 75). Null space is increasingly used in explanation of operations for projective geometry and image processing (Faugeras and Luong 2001; Hartley and Zisserman 2003).

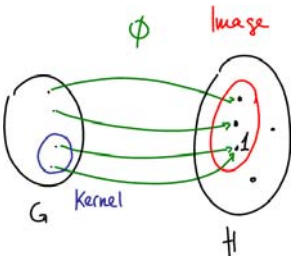


Figure 42: The Image and Kernel of G under Φ

kernels are general constructions which measure the failure of a homomorphism or function to be injective.[wikipediaentry kernel]

Image and kernel can be used to identify different types of morphism:

If $\Phi : G \rightarrow H$ is an

Epimorphism	*<-> Im (Φ) = H*	*Φ is surjective*
Monomorphism	*<-> Ker (Φ) = 1*	*Φ is injective*
Isomorphism	*<-> Ker (Φ) = 1 and Im (Φ) = H*	*Φ is bijective*

7. CATEGORIES

Constructing computational models reduce the complexity of the world to constants and variables, procedures and functions. The conceptual diversity can be reduced further and procedures, functions and constants all seen as functions with a different number of arguments; constants are functions with no argument(Bird 1998). This simplification allows a view of operations that can describe the semantics of operations without resorting to representations: we describe the semantics of the operations by formulae without reference to the elements they are applied to. This approach is typical for mathematical category theory(Pitt 1985; Barr and Wells 1990; Asperti and Longo 1991; Walters 1991). Category theory is a generalization of (universal) algebra and shares with algebra many concepts – often with slightly different meaning.

Category theory is an application of concepts of algebra to algebras themselves. It is not related to the category theory of cognitive science and ontology, where classes (categories) of similar objects are formed(Rosch 1973; Rosch 1978).

7.1 A CATEGORY AS AN ALGEBRA OVER FUNCTIONS

Category theory deals with categories, which consist of arrows connecting objects. An arrow *f* maps from the domain *A* to the codomain *B: f : A -> B*. The most intuitive example for a category is the category of sets, where functions are the arrows and sets are the objects, but category theory is much more general. Category theory is a simple, general algebra of arrows and the operation of interest is the composition of two arrows. For each domain there is a unit, which maps every element to itself. For functions this is constant function identity *id* that does nothing: *for all x: id x = x.*

A category is "a collection of algebraic systems and their morphisms" (Mac Lane and Birkhoff 1991, 129)

The primary operation in a category is the composition of arrows. It is written as '.' and must be associative; this makes parentheses unnecessary as *(a.b).c= a.(b.c) = a.b.c.* Composition

is partial and only defined when the domains map: *f: A -> B, g : B -> C; (g.f) : A -> C*. Function composition follows the rule:

for all f, all g, all x: (f.g) x = f (g (x)).

Note that function composition for the category of sets and functions (see chapter 4) the above formula is second order, quantifying over all functions *f* and *g*. Categories are like groups: they have a single operation that is associative and has a unit, but not necessarily an inverse (or like a semi-group with a unit).

Categories treat algebras the same way than algebras deal with entities(Frank 1999).

Comment on notation: When taking a functional point of view, function application is the most common operation: *f* is applied to *x.* In analysis multiplication is the most common operation *ab* means *a * b* and function application is written as *f (x).* In a functional context, where function application is the most common operation, we just write *f x* to indicate the application of *f* to *x*—no parenthesis required (parentheses are used for grouping as usual). This is not used consistently in this text; when it is convenient, then the traditional *f(x)* notation with parenthesis is used.

Category <·,id>	
	dot, (·) :: f -> f -> f
	condition: a . b defined only for codomain b = domain a
	id ::o -> f (there is an identity for each object type o)
	domain :: f -> o
	codomain :: f -> o
associative:	*(a.b).c= a.(b.c) = a.b.c.*
unit:	*id(codomain f) . f = f*

Category theory gives us a high level, abstract viewpoint: instead of discussing the properties of elements we directly address the properties of the operations. This corresponds to the interest in geography, where the discussion concentrates on processes that occur in space, not on the collection of locations and properties of spatial objects(Abler, Adams et al. 1971; Frank 1999).

In the category of sets, composition combines functions like ordinary operations (e.g., addition) combine numbers.

The properties of operations are described—as far as practical—without reference to the elements the functions are applied to. To state that two functions *op* and *inv* are the *inverse* of each other, we simply write *op . inv = id*. For the function increment *inc* and its inverse decrement *dec* the composition is the identity function: *dec.inc = id.* To state that a function can be applied any number of times and producing the same result as a

single application—we say the function is *idempotent*—one writes *op. op = op*.

A categorical viewpoint demonstrates that semantics of operations are independent of the representations they are applied to. A '*pointless'* definition is independent of the representation and this is documented by the absence of the elements (sometimes called *points*) the functions are applied on in the definition.

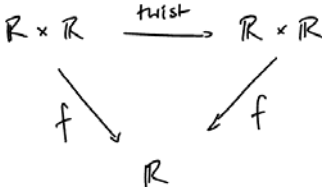


Figure 43: Commutative law (twist (a,b) -> (b,a))

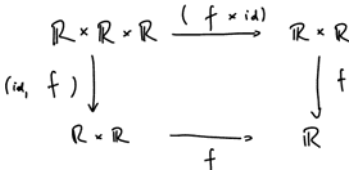


Figure 44: Associative law

7.2 COMMUTATIVE DIAGRAMS EXPRESS AXIOMS

The axioms for a group can be expressed as commutative diagrams. The commutative law *(a + b = b + a)* is shown in Figure 43: Commutative la, where the function *twist :: A x B -> B x A, twist (a,b) = (b,a)*. The associative law gives *(a + (b + c)) =((a + b) + c).*

To show these axioms as commutative diagrams stresses that they are generally applicable, for many arrows and domains, not only the category of sets with functions; the arrow *f* can be +, but could be some other operation that follows these laws.

7.3 CATEGORIES CONTRIBUTE TO THE UNIFICATION

Category theory is the generalization that allows us to bring together different parts of mathematics and identify the commonality. Most fields in mathematics deal with a category, as shown in the following table(based on Asperti and Longo 1991, 4); category theory establishes the connections between them.

Category	objects	Morphism	Part in this book
Set	sets	functions	Chapter 6 Measurements
Top	topological spaces	continuous functions	Part 7
Vect	vector spaces	linear transformations	Part 3: Space time, Part 6: Projective Space
Grp	groups	group homomorphism	
PO	partial ordered sets	monotone functions	
Graphs	edges and nodes	graph morphism	Part 8
Rel	relations	join	Part 5: DB

8. REPRESENTATION AS MAPPINGS: PRACTICAL PROBLEMS

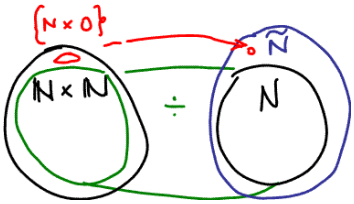


Figure 46: Extension of Natural Numbers to make division total

Many common problems in Computer Science and GIS can be analyzed in terms of properties of operations and mappings from real world to computer representations. For example, that the division is not a total function (no division by 0!) is well-known, but even commercial programs fail because programmers forgot to check for this case(Goldenhuber 1997). A systematic solution for all such cases of non-total functions is the extension of the codomain of the division with an additional value '*not a number*' (NAN in the ISO standard for numeric computation[ref]) or similar, to which all division by *0* are mapped. The new function is then total! We will later call such morphisms *functor*s (see next chapter 6).

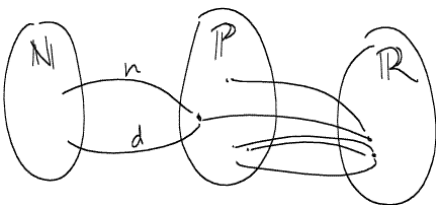


Figure 47: Construction of rational numbers with a surjective function to map to the rational numbers

8.1 TOO MANY REPRESENTATIONS: RATIONAL NUMBERS

Rational numbers can be represented as pairs of integers, like (*1/2, 2/4, 3/4*, etc.), but many pairs, for example, the pairs *2/4* and *1/2,* are the same value. Generally, all values *i*n/i*d* for any *i* are equivalent. We select the value *n/d* as the representative of the equivalence class. This defines a surjective function from pairs *P* to the (reduced) rational numbers *R*. This function has no inverse, given a rational number *3/4*, we can not determine if this was originally *3/4, 6/8, 9/12*, etc.

Solution for too many equivalent solutions: select a canonical value to represent each equivalence class.

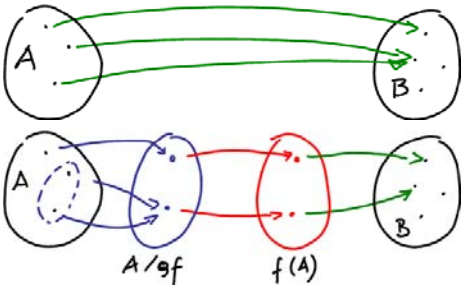


Figure 48: Canonical factorization (after(Gill 1976p.56))

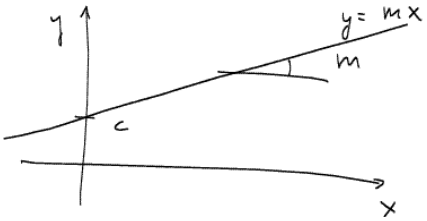


*Figure 49: A straight line represented as y=m*x+c*

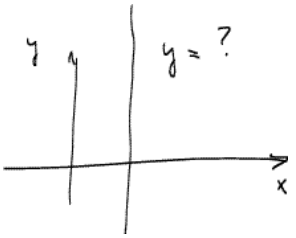


Figure 50: A vertical line is not representable as a pair m and c

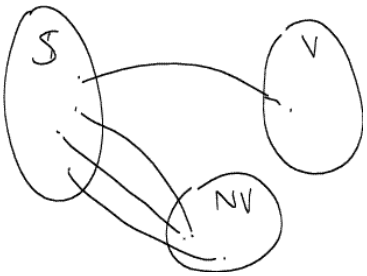


Figure 51: Redundancy allows separation between valid and non-valid tokens

Algebras describe abstract structure, which can be preserved across transformations between different representations.

A general solution is found through canonical factorization. Given a function *f: A -> B*, the equivalence kernel of *f* is a relation *ρ (a, b),* which is true when $f(a_i) = f(a_j)$. *ρ* is an equivalence relation and induces a partition A/ *ρ*, where each equivalence class consists of all elements whose image is a given element in the range of *f* (Gill 1976p. 57)

8.2 PARTIAL REPRESENTATION: STRAIGTH LINES

Representations that cannot represent all values of interest cause difficulties. Here a geometric example: straight lines can be represented as functions *f (x) = y*, with *y=m*x+c*, which suggest a representation of a straight line as pair of values *m* and *c*. This mapping from straight lines to pairs of reals is partial, because lines parallel to the *y*-axis (vertical lines) have no representation in this form. We will later give a different representation for straight lines (chapter 19), which has for each straight line multiple representations, that is, a case of 'too many representations', but at least can represent all straight lines.

8.3 REDUNDANCY

Representations that allow many more tokens than are needed to represent the intended values can be used to guard against errors. A rule that defines the unused tokens as illegal allows differentiating between intentionally produced legal tokens and erroneous tokens (Figure 51: Redundancy allows separation between valid and non-valid tokens). Errors in the transmission can be detected if they result in an erroneous token. A typical example is the introduction of a parity bit, to guard against transmission errors.

9. CONCLUSIONS

Formal methods rely on the manipulation of symbols according to some rules, which are written as sequence of symbols and called programs. The logical approach shows how true statements are transformed to other true statements; the algebraic viewpoint stresses the general rules of such transformations.

Formal systems show how to translate one representation into another one, preserving properties of interest. Algebras are descriptions of the structure of formal systems and define the concept of structure. It is possible to understand the production rules in the language definitions as functions (morphism)(Ehrich, Gogolla et al. 1989; Loeckx, Ehrich et al. 1996). This is useful as

it shifts the focus from the (often infinite) set of sentences in a language to the finite set of production rules and leads to conclusions about a language based on properties of the production rules.

Morphisms are used here to "construct from simple parts complex systems"; we have seen how length of string is a morphism from strings to integers, which maps string concatenation to addition. In the next chapter, we will generalize this notion, using the concept of a functor from category theory.

Category theory is an abstract treatment, where we concentrate on the operations, independent of the representation. It is useful when we have to bring together in the GIS different parts of mathematics, e.g. set theory, geometry, topology and relations. It is increasingly used in computer science, for example in image processing.

REVIEW QUESTIONS

- What is an algebra? What does it consist of? Give an example.
- Explain what a total function is. Give examples. Why is this important for programming?
- What is a homomorphism? Give three examples.
- What is the connection between Boolean Algebra and Boolean Logic?
- What is a category? Why are we interested in it?
- What is the '.' (dot) operation? Explain in a formula in a style you are familiar with.
- Why are isomorphisms practical? Why can we say that they are 'transparent'?

Chapter 6 OBSERVATIONS PRODUCE MEASUREMENTS

Figure 52: A surveyor observes a distance and produces a measurement

GIS store observations of the outside reality. We will use the notion *measurement* for the representation of the results of such observations in a general sense. Measurements can be the result of surveying operations with instruments, counts resulting from statistics or other observations of physical properties.

We observe reality. Observations can be the color of a field, the height of a point or the force of gravity. The result of an observation is expressed as a value on the appropriate measurement scale; for different observations different measurement scales apply: color is recorded as a value like 'red' or an RGB triple (red, green, blue intensity), whereas the height of a point is *324.4 m* above mean sea level or the force of gravity is *9.9413487 mgal*. Typed functions then connect these values to other values on different measurement scales; for example, the area of a rectangle is calculated as the product of two length values and the result is expressed as square meters.

The representation for measurements is produced by a language (see chapter 4). The results of observations are typed expressions on some measurement scale, which are algebras, understood best as sets of operations that are possible with these values such that operations with the values relate to operations in reality. Measurement scales determine, for example, what statistical operations are meaningfully applied to some observations. Scales of measurements are—in the terminology of programming languages—(abstract data) types.

In this chapter we will study functors, which map between types. Functors transform types, preserving the intentions, the semantics, or—technically—the algebraic structure. These three notions are used here as synonyms. Transformation of types by functors is different than the 'type cast' operations, which change just the type and do not preserve the algebraic structure(Stroustrup 1991).

Measurement units are functors, they map numbers to measurements, and the operations that we want to apply to measurements, to operations on the numbers.

1. REPRESENTATION USING A LANGUAGE

The values must be represented. A formal language, for example the language of decimal point numbers, produces distinct values, but the representation and the type is not the same: one kind of

representation can be used to represent values that have different semantics and allow different operations. In many currently used programming languages, type and representation is incorrectly equated.

Soil types:
sand = 1,
gravel = 2,
podsol = 3,

For example, the representations of soil observations are made on a scale of sand, gravel, podsol, etc. These nominal values may be represented with numbers, but this does not imply that numerical calculations make sense. It is not meaningful, to sum such numbers, for example calculations that the average between podsol and sand is gravel are utter nonsense! This example shows that measurement scales are not just representations but algebras.

2. ENTITIES AND VALUES

An entity is anything conceptualized as having a distinct existence(Zehnder 1998).

Observations presuppose that we observe something as distinct from other things. We will call these things 'entities'. Anything for which we assume a distinct existence and durability in time is an entity. A first type of entity is a point in space and time, for which different observations are possible.

An observation connects an entity with a measurement value.

The result of an observation is a measurement, which is a value selected from a collection of values. For example, observations of color are sometimes selected from the set of values *red*, *yellow*, *blue*, etc. or the observation of today's temperature in ℃ (Celsius) results in the value 13, a distinct value of type integer, that is, from the set of values 0, 1, 2… , etc. Some people describe the temperature more precisely as 13.5 °C, a value from floating point numbers.

3. TYPES OF MEASUREMENTS

The set of values from which an observation selects one is a type(Cardelli 1997). Different observations of the same kind all result in values of the same type—but distinct values. The temperature yesterday was perhaps only 10°C, which is a different value but of the same type as today's temperature. The distinction of types makes it possible to guard against nonsense operations, like the one shown in Figure 53.

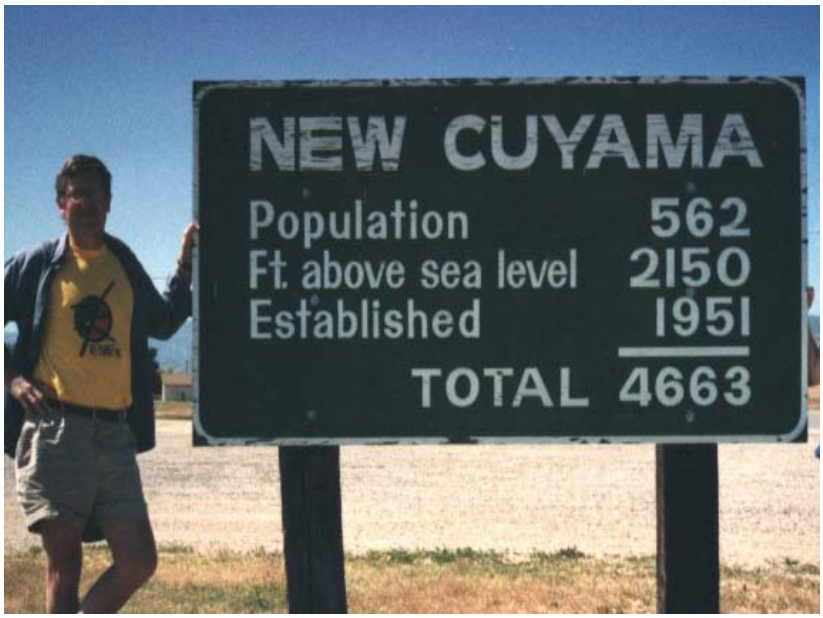


Figure 53: The description of New Cuyama, California (Mike Goodchild holding, picture by Helen Couclelis)

Measurements are not just representations, because representations alone would not be typed. Representations alone do not have an algebraic structure. A typed formalization allows automatic type checking of all formulae; this increases our confidence in the formalization—many common mistakes in human reasoning are discovered by type checking. Type checking in a formal language is similar to the checking of dimensions for formulae in physics. Most students of physics learn to control their formulae by checking that they are correct for the dimensions.

Example:

*s = v * t, where*
v velocity in m/sec
t time in sec
s distance in m
*[s] = [v * t] = [v] * [t] -> m = m/sec * sec*

Types represent a part of the structure of reality(Asperti and Longo 1991).

Such formulae, connecting measurements of one type with measurements of another type are the fabric that makes an information system! They are expressions of the semantics of the corresponding measurement types. Conversions of measurement units are not changes in types: the same operations apply to the length measured in m or in feet; it is only a conversion of the numerical values that represent the multiplicity of the unit to achieve the desired value.

Physical dimensions are different types but different measurement units are not different types.

4. FUNCTORS

Measurements are expressed on scales appropriate for the observation and preserving structures that we assume to exist in

reality. Most of the time we forget the algebraic properties of the measurements and operate with them as if they were just ordinary numbers without the special properties of measurements. This custom to calculate with the numeric values of measurements is most often justified (Figure 53 shows the exceptions) because the algebraic structure of the measurement scale and the numbers are the same. We can see the measurements scales as *functors* that construct new algebraic systems from the given ones.

"many constructions of a new algebraic system from a given one also construct suitable morphism of the new algebraic system from morphism between the given ones. These constructions will be called 'functors' when they preserve identity morphism and compositive morphism."(Mac Lane and Birkhoff 1991, p. 131).

4.1 DEFINITION OF FUNCTOR

Given two categories *A* and *B* and two objects A_1 and A_2 in *A* then a functor *F* from *A* to *B* consists of functions

F :: obj A -> obj B (i.e., it maps a function in A to a function in B),

and for each pair of objects A_1, A_2 of *A*, with

g :: A_1 -> A_2, F (g) :: F (A_1) -> F (A_2),

satisfying

F (id) = id,
F (k.l) = F (k) . F (l) (when k :: A_2 -> A_3 and l :: A_1 -> A_2) (Walters 1991, 93)

4.2 FUNCTORS CONSTRUCT TYPES

Consider vectors with 3 elements as a functor F :: Real -> Vec3. The unit 0 is mapped to <0,0,0>, addition maps to pointwise addition of the elements: <a,b,c> + <d,e,f> =<(a+d), (b+e), (c+f)>. You can immediately see that the group properties are preserved; e.g. v + 0 = <a,b,c> + <0,0,0> = <a,b,c> etc.

Pointwise application of a function: Apply the function to each element.

Dimensioned measurements are types, constructed by a functor. We will use the name of the measurement unit for these functors (e.g., meter) *meter :: R -> L*. The functor *meter* takes a (numeric) domain *R* and constructs the domain of 'length measurements' *L*, e.g., real numbers -> length in real numbers (i.e., *R -> Length R*); the same functor meter takes a function *add:: R x R -> R*, to *meter (add) :: Length R -> Length R* and the diagram in Figure 55 commutes.

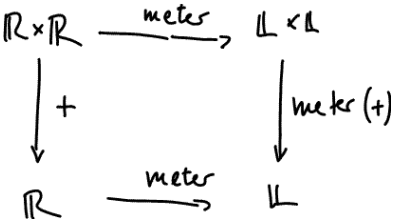


Figure 55: Functor meter

Categories generalize algebras; Functors generalize morphism.

5. MEASUREMENT SCALES

Different observations result in different kinds of values: the determination whether a student passes a course is a Boolean value (*True* or *False*), the students grade is on a scale *A, B, C, D,* and *F*, today's temperature is *13 °C* and my height is *182 cm*. Stevens (1946) identified differences in the way measurements must be treated; he called them measurement scales. In our terminology, a measurement scale is an algebra, which defines

what operations can be applied. The operations applicable determine then, for example, what statistical operations are possible, because statistical operations depend on basic arithmetic operations.

Traditionally four measurement scales are differentiated and correspond to algebraic structures that are well-known(Stevens 1946):

- Nominal -> equality
- Ordinal -> order
- Interval -> 1D affine space
- Ratio -> field

Arguments to consider absolute, logarithmic scale, count, and cyclic scale as measurement scales have been published(Chrisman 1975; Frank 1994; Fonseca, Egenhofer et al. 2002), but no agreement has been reached yet.

Measurement scales are mostly discussed in cartography and statistics. In cartography they help to select an appropriate graphical representation for a set of observations: the graphical properties of the representation must have the same algebraic structure as the value to depict graphically and the transformation from an internal representation to a graphical representation must preserve this algebraic structure (Bertin 1977; Chrisman 1997, 13). Increasingly other applications find the concept of measurement scales useful.

6. NOMINAL SCALE

The least structured measurement scale is a nominal scale: the result of an observation is a value, of which we can only say if it is the same or different from another one. Examples:

```
Soil types: gravel, sand, podsol, etc.
Land use classes: agricultural, residential, forest
Names of people: Peter, Fritz, John (names in general)
```

A special case of a nominal scale is the two truth values *True* and *False* encountered before.

The algebraic structure is the algebra of equality. It has two binary operations, namely a test for equality and a test for not-equality that result in a Boolean value, and a single axiom, which says that not equal is the same as not-equality. The equality relation must be transitive, symmetric, and reflexive:

Equality	
inverse	*not (=) = ≠*

transitivity	*(a==b) && (b==c) => a == c*
commutative	*(a==b) => (b==a)*
reflexive	*(a==a).*

7. ORDINAL SCALE

Observation can result in values that are ordered; one can tell if a value is more or less than another value, but not how much more. Examples: Size of T-shirts: *Small, Medium, Large, XLarge*. Grades in School: *A, B, C, D,* and *F*. In each case, we know that *Large* < *XLarge* or *A* > *C*, etc.

One can compare two values on an ordinal scale and determine which one is bigger, but it is not possible to say whether the difference between two values is the same than the difference between two other values. It is true that an *A* is the better grade than a *B*, but to state that the difference between an *A* and a *B* grade is the same than the difference between *B* and *C* is for most tests nonsense.

7.1 TOTAL ORDER

Values on the ordinal scale are supporting the operations of the nominal scale, that is, we can differentiate two values. Order is imposed on a collection of values by a relation (<=) that is transitive, anti-symmetric, and reflexive (compare: equality is transitive, symmetric, and reflexive). Other operations, like <, >=, >, are derived and need not be defined individually.

Total Order	
transitive	*(A <= B) and (B <= C) => A <= C*
anti-symmetric	*(A <= B) and (B <= A) => A == B*
reflexive	*(A <= A)*
totality	*(A <= B) or (B <= A)*

For ordinal scales, the maximum or the minimum value from two arguments can be computed.

max (a,b) = if a > b then a else b
min (a,b) = if a < b then a else b

In bounded data types—and all data types representable in a computer are bounded—the maximum value is the unit for *min* and the minimum value is the unit for *max*:

max (minVal, a) = a
max (maxVal, a) = a.

7.2 LEXICOGRAPHIC ORDER

It is common to impose on nominal scales that are not ordinarily ordered an arbitrary ordering, called lexicographic order.

Assuming that the letters of the alphabet—which by themselves are also on a nominal, unordered scale—can be arranged in an order, namely the order of the alphabet, one can deduce an order relation between any two names. This is convenient and can speed up search procedures considerably. Imagine searching for a name in a telephone directory that was not arranged in alphabetical order! The same principle can be applied to many other values that do not have a 'natural' order. These are tricks to improve performance and do not correspond to an order in reality! From the fact that "Frank" is ordered before "Heinrich" one must not conclude anything about the properties of the two families (Figure 56).

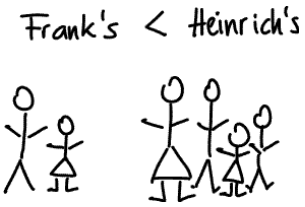


Figure 56: Two families

Beware of different alphabets and orders for different languages. Austria adds the Umlaut Ä, Ö, and Ü at the end of the alphabet (Swiss translate them as Ae, Oe, and Ue and order them at the corresponding place). A Spaniard uses an alphabetical order A, B, … L, LL, M, N, Ñ, O, P, Q, R, RR … etc. Can you imagine an "alphabetic order" for Chinese characters?

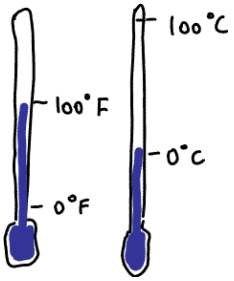


Figure 57: Different zero's

8. INTERVAL SCALE

The interval scale is the scale represented with numbers, for which the computation of a difference is meaningful. On the interval scale, no absolute zero is defined (Figure 57). The most common examples are temperatures expressed on conventional scales (Centigrade or Fahrenheit). One can calculate differences: the difference between days with *20* and *25 °C* is the same as the difference between *10* and *15 °C*. However, the difference of *5* °C is not the same as a day with *5 °C* temperature. The value of a difference is expressed on a ratio scale (next section) and the operation difference has an inverse.

Mathematically, an interval scale is a *1*-dimensional affine space, where we have the numbers of the interval scale *I* and real numbers *R* and the operations *diff* and *plus* that map from the interval scale to the real numbers (Mac Lane and Birkhoff 1991, 564):

diff (a,b) = r => plus (a,r) = b
diff (a,a) = 0, plus (a, 0) = a
plus (plus (a,r), p)) = plus (a, r + p)

This is the foundation for statistical operations with interval data: we take the difference to some arbitrary base (for example the value zero) and then compute with these ratio values as usual. The result of the average must then be added to the base. The

base may be the arbitrary zero of the scale diff and plus are then not changing the numeric value, but just the type. They convert the ration type in which differences are expressed back into the interval type.

9. RATIO SCALE

If there is an absolute zero—determined by properties of the phenomenon, not an arbitrary selected point like the freezing of water for 0°C, then we have a ratio scale. On a ratio scale, it is possible, to compare two values and say that $20 are twice as much as $10, which would be nonsensical for interval scales: a day with 15°C noon temperature is not half as warm as a day with 30°C at noon! For the temperature scale, 0°K (*Kelvin)* is an absolute zero. Length is measured on a ratio scale—the zero is the distance from a point to itself—but measurements for the length may differ when users use different measurement units: the 0 is fixed on the scale, but not the 1. For example, a sheet of A4 size is 210 mm or 8.27 inches (Figure 58).

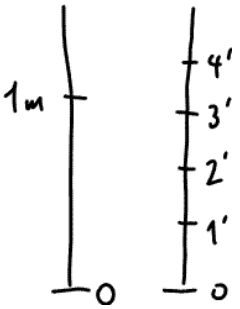


Figure 58: Measuring with meter and feet

The results of measurements expressed on the ratio scale leads to the algebraic structure "field", with the operations + and * ; inverses for both; and a defined *zero* and *one* as units for the two operations. Fields are a special case of rings, which are introduced first:

9.1 ALGEBRAIC STRUCTURE RING AND FIELD

We carry out ordinary arithmetic calculations with integers, (approximations) to real numbers or with fractions and are assuming more algebraic structure than just a group (see chapter 5). We use two operations (+, *) and each has a unit (*zero* and *one*, for + and *, respectively). This structure is called a ring, which is a an Abelian (commutative) group <R, +, 0> with an additional operation and a unit (these two form a monoid <R, *, 1>). The two operations are connected by the distributive axiom.

Ring <R; +,, 0>*	*Abelian group for <R,+,0>*
associative	*a* (b * c) = (a * b) * c*
distributive	*a * (b + c) = a * b + a * c*

The identity for the multiplication is usually denoted by *1*:

identity $1 * a = a * 1 = a,$

and it may be commutative

$a * b = b * a,$

but not all rings are commutative (e.g. the ring of matrices where *A B /= B A*). An *integral domain* is a commutative ring with

multiplicative identity that satisfies the cancellation law(Gill 1976, 288-289):

$a * b = a * c \quad \Rightarrow b = c$
$b * a = c * a \quad \Rightarrow b = c.$

A ring that satisfies the cancellation law has no divisors of zero, that is, it has no non-zero elements a and b such that $a* b = 0$.

*Field <F, +, -, *, $^{-1}$, 0, 1>*	*Commutative Ring*
*Inverse for *:*	$a * (a^{-1}) = 1 \quad$ *for* $a \neq 0$

A *field* is a ring with an inverse for the multiplication with a corresponding axiom.

$(a*a^{-1}) = 1 \quad$ *for* $b \neq 0$

Rational numbers (fractions) are another example for a field. The real numbers form a field, but we always use finite approximations, which only approximate these axioms:

$a * (a^{-1}) \sim\sim 1.$

10. Other Scales of Measurements

Most observations produce values from one of the above scales, but a few other are sometimes used: absolute scale, counts and cyclic scale are discussed here.

10.1 Absolute scale

Probability is measured on a scale *0* to *1*. This is more determined than a ratio scale, because not only the *0* is fixed, but also the *1*. There are no transformations possible and necessary.

10.2 Count

Counting results in positive integers. There is a zero and there is a one—which makes it an absolute scale expressed in integers, but the ratio between two counts is expressed as a fraction (remember: fractions are a field). The difference between two counts is again a count—which shows a difference from interval scales; the ratio of two counts is not a count—which shows a difference to the ratio scale. Therefore counts are a separate scale.

There is a conceptual difficulty with results from statistic. We expect generally that the average is expressed on the same measurement scale as the original observations. A transportation authority observes the number of persons per car: The values are expressed as positive integers, i.e. count, but the average will be a figure like *1.3* persons/car—which is expressed as a real. This is not a contradiction, as the value is not of type persons, but persons per car.

10.3 CYCLIC SCALE

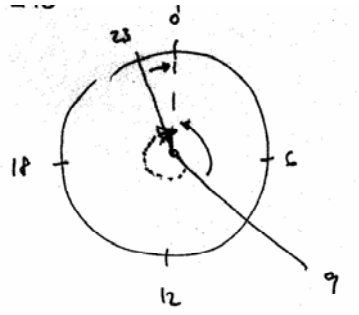


Figure 59: Time Measurements on Cyclic Scale

The results of observations of regularly *repeated* properties result in yet a different measurement scale: the measure of an angle, the time of day or the date in a year are expressed on a cyclic scale. It is difficult to say if *9 a.m.* is before or after midnight, *9* hours after midnight comes *9 a.m.*, but *15* hours after *9 a.m.* comes midnight again. It is before and after midnight and order as defined for a linear scale is meaningless. It is a convention to say that *9.a.m.* is after midnight and *11 p.m.* is before midnight, because *9 a.m.* is closer to the midnight before, whereas *11 p.m.* is closer to the midnight afterwards. The same applies for angles and other cyclic measurements(Frank 1994).

This is an example where we have multiple representations for the same value: the angle expressed as *20°, 380°, 740°,* etc. are all the same. On the regular *12* hour dial, *9 a.m.* and *9 p.m.* (*21 h*) are the same. To make processing simpler, we select one preferred representation for each value, among the many, and call this 'canonical representation' (see before chapter 5.8xx).

Canonical representation is the selection of one element of an equivalence class to represent the class.

The concept of Image and Kernel (see chapter 5.6xx) can be applied here. The reduction of a large set of values *H* to a smaller (canonical) one *G* can be seen as a morphism *ρ: G -> H*. In our example of angles, *G* is the real number line and *H* is the interval of (*0..2π*); the image of *G* is all of *H* (Figure 60); the values *n*2*π* for all integers *n* is the kernel of this mapping. The morphism *δ* collapses the real numbers to the interval [*0..2π*].

11. MEASUREMENT UNITS

*Measurement = unit * value*

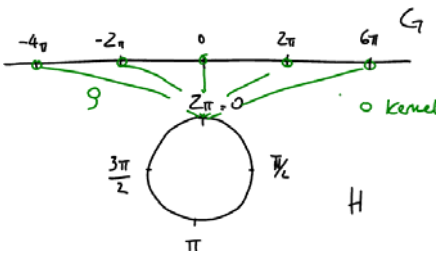


Figure 60: The canonical mapping δ of the real numbers to angles

Measurements describe the quantity or intensity of some properties at a given point in comparison with a standard quantity. The same observation process yields different values if applied at different points in time or space. The observed measurements are usually proportional to the intensity of the property at that point. The results of observations become comparable, if they are each compared with a selected standard value. Well-known is the former meter standard, defined as the distance between two marks on a physical object manufactured from platinum, which was kept in Paris. It is superseded today by a new definition using a physical process that can be reproduced in any location. The current definition is stating that a meter is the 1/299 792 458 part of the distance light travels in the vacuum in a second(Kahmen 1993). The reference point for

the Celsius scale of temperature is the temperature of melting ice at 0ºC and the reference point for 100°C is the temperature of boiling water and similar definitions exist for other standard quantities.

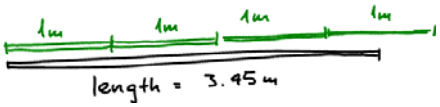


Figure 61: Figure standard rod used to measure

The combination of a measurement unit with an observation value is a mapping between two types, namely from numbers to measurements. The value *3.2* is mapped to *3.2 m* — *3.2* is multiplied with the unit *1 m*. The measurement unit is a functor, which converts a number to an element of domain with a dimension and unit (Figure 55) and ordinary algebraic operations apply to these quantities with units:

*3 * (3.2 m) = (3 * 3.2) m*

11.1 Calibration

Observation systems are calibrated by comparing their results with the standard. The raw measurement results are then converted with some formulae to yield a measurement value, expressed as a quantity times a unit, *3 m*, *517* days or *21*ºC. Different observation processes that measure the same physical dimension (e.g., length, time) are brought to a common base.

11.2 Base units

The selection of base units is arbitrary. People use convenient units, based on the cultural environment and such that the numerical values for many measurements are small but sufficient to differentiate relevant differences: we use *mm* for table-top items, meters for apartments and gardens, *km* for geography, etc. Conversions are necessary during input and output of values, because a single internal computations in floating point numbers is sufficient and only a difference in the exponent results from different units ($3.5 * 10^3$ *m* = $3.5 * 10^6$ *mm*).

The Système International d'Unités (SI) is founded on seven *SI* base units for seven mutually independent base quantities (Table 1). This system of units superseded the previously used *cgs*-system, where the units where centimeter, kilogram, and second. For example, the unit of gravity in the cgs-system was *Gal*, named after Galileo (*1 Gal = 1 cm* s^{-2}), newer books refer to the SI unit as *m* s^{-2}.

Table 1: SI units

The mutually independent SI base quantities:
meter (length) m
kilogram (mass) kg
second (time) s
ampere (electric current) A
Kelvin (thermodynamic temperature) K
mole (amount of substance) mol
candela (luminous intensity) cd

USA and some other English speaking countries use traditional units like *feet* and *pounds*. Additional confusion can result from different definitions for different countries: imperial (English) and U.S. definitions, sometimes with regional variants for surveyors, abound. Units may also differ, depending what is measured: fluid ounces and troy ounces (for gold) are different. A rumor has it that the loss of the probe to the Mars—a costly NASA space exploration mission—was due to passing a value from one program to another where the one assumed SI units (i.e., meters) and the other assumed traditional units (i.e., feet) for the height above ground.

11.3 Conversion of Measurements

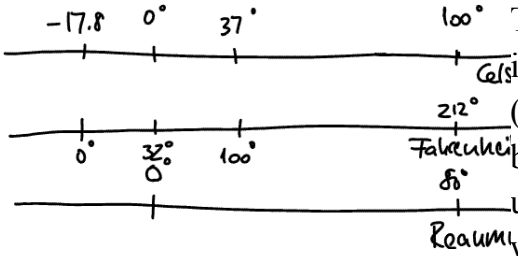


Figure 62: Three scales for temperature

Conversion is not a change of type

Theoretically the conversion of one measurement unit to another is usually a linear formula, like the conversion of inch to *mm* (multiply by 25.4) or ºC to ºR. The general case is the conversion between two measurement scales on interval scales, where the units and the zero point are different (Figure 62). This is, as we will see later (see chapter 10), an affine transformation in *1*-dimensional space.

Example: The conversion between Centigrade and Fahrenheit, which is used in the USA and Jamaica. The definition of the Fahrenheit scale is set today to *0 ºF = 32 ºC* and *212º F* is *100º C,* which results in convenient conversion formulae. The original concept was to fix 0º to the freezing point of alcohol (17.8º C) and 100°F to the temperature of the human body (37 °C). For conversion use:

$$[°F] = [°C] \cdot 9/5 + 32$$
$$[°C] = ([°F] - 32) \cdot 5/9$$

In general, a *1*-dimensional affine transformation is determined by two parameters (Figure 63) (Mac Lane and Birkhoff 1991, 561):

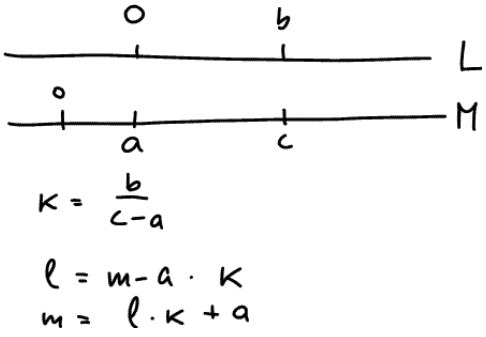


Figure 63: 1-dimensional affine transformation

12. Operations on Measurements

The operations applicable to the numerical values representing measurements on the ratio scale are those of a field, which are the ordinary arithmetic operation. Not all of these operations are meaningful. Because measurements types are functors, measurements can be added and subtracted and multiplied or divided by a scalar value (i.e., a real number with no measurement type). Other combinations, e.g., the multiplication of two measurements give as a result another measurement type

(see next section): the multiplication of two length values resulting in an area value, not a length. Measurements form an algebraic structure, which is called a module (see later 9.5xx)

*Measurements <M, S, +, -, 0, 1, *, 0m, 1m, />*	
Addition, Subtraction	
Scalar Multiplication	*s1 * m + s2 * m = (s1 + s2) * m*
	*m / s = 1/s * s*

13. Combinations of Measurements

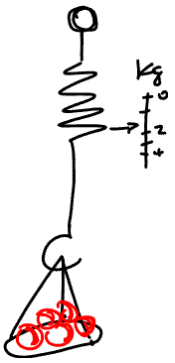


Figure 64: The spring based scale used on markets all over the world

Measurement instruments observe some easy to detect quantity—for example, an electric current—which is in some direct relationship with the quantity of interest—e.g., the amount of light. The instrument then includes an analog or discrete computation to compute the value of interest. For example a balance measures the elongation of a string under, which is proportional to the weight attached to the hook (Figure 64). The computation of an area results from observation of two length measurements. The area value is the product of the two lengths values, but its type is a different one:

```
area :: Length -> Length -> Area
```

This approach requires the definition of suitable types and coding of the standard formulae and will guard against confusion between units and dimensions.

14. Observation Error

All observations are imperfect realizations and have some error. This is in the limit a consequence of Heisenberg's uncertainty principle, but most practical observations are far less precise than the uncertainty principle would permit. Measurements more accurate than 1 part per million (ppm) are generally difficult. Distance measurements with an error of less than 1 millimeter per kilometer are demanding, but few centimeters per kilometer are standard performance of surveyors today. The best observations are for time intervals, where 10^{-15} is achieved, but the theoretical limit would be 10^{-23}, still 100 million times less!

Parts of the error of real observations are the result of random effects and can be modeled statistically. Surveyors report measured coordinates often with the associated standard deviation, which represents—with some reasonable assumption—an interval with 68% chance to contain the true value. Errors propagate through the computation. The Gaussian law of error propagation approximates the propagation of

random error; it says that the error propagates with the first derivation of the function of interest. Given a value $a = f(b,c)$ and random errors for b and c estimated as e_b and e_c (standard deviations), then the error on a is following Gauss:

$$e_a = sqrt(df/db * e_b^2 + df/dc * e_c^2).$$

In this book, errors are not in the focus and all quantities are assumed to be 'perfect' knowledge.

15. Abuse of Numeric Scales

Measurement scales determine what operations are possible with the values; they determine, among other things, what statistical operations are appropriate. Unfortunately, it is customary to express values on a nominal or ordinal scale with integers or reals—and it is then technically feasible to calculate with values that do not have the required properties for these calculations.

There are numerous examples for abuse of ordered scales, representing them with numerical values and then compute averages. Common is the computation of average grades in school. Grades are expressed on an ordinal scale; a difference between two grades is not a defined quantity. I do not believe that the difference in knowledge of a student between a grade of A and B and the difference between grade B and C is the same—but this is assumed to calculate the average. This method is used because we do not have a better solution to arrive at a fair and equitable determination of final grades in a class where multiple exams where taken, but we should be aware of the limitation.

16. Conclusion

Measurements describe observation of a physical dimension; different physical dimensions are different types and cannot be mixed, the logic of a typed language helps to avoid nonsensical operations as adding a date and a length (Figure 53). Internally all measurements of one physical dimension can be expressed with the same units, conversions are necessary for input and output.

Measurements are expressed on scales of measurements, which each represent an algebra that determines what operations are possible with measurements of this kind.

REVIEW QUESTIONS

- What is wrong with the panel in Figure 53? Would a type language discover the problem (e.g., Pascal)?
- What is a canonical representation? Why is it useful? Give a practical example from real life.
- Define Group, Ring, and Field. Give axioms.
- What are scales of measurements? What are the classical measurement scales?
- Explain the concept of functor? How is it applied to measurements?
- What is a partial order? Give an example.

PART THREE SPACE AND TIME

Position in space and time are fundamental for a GIS. They allow connecting other observations to locations in space and time. Measurements of length and duration determine relations between points in space and time. But not for all applications of GIS the metric properties of space are crucial and other aspects are more. For example, to determine a path in a network, connections between the nodes are crucial and more important than the distances. A GIS must connect different conceptualizations of space and allow an integrated analysis of facts related to them(Couclelis and Gale 1986; Frank and Mark 1991; Mark and Frank 1991).

In this part we start the discussion of geometry following the approach by Felix Klein(Klein 1872). We will not follow the customary route of separating geometry by dimension, discussing *1-, 2-, and 3-dimensions* in turn, but differentiate types of geometries independent of dimension, which have different approaches to discretization and abstraction of continuous space and show the transformations and invariants in each. Different aspects of space lead to different conceptualizations of geometry and geometric properties. Quantitative approaches in geography are often based on transformations of space, such that certain relations are expressed more directly(Tobler 1961). For example, the map of a city is transformed such that distance on the map directly represents travel time to the center (fig. xx).

Geometry: properties which remain invariant under a group of transformations.

Geometry in gymnasium deals with geometric constructions: geometric elements situated in space gives structure to space and allow operations, which result in other geometric elements(Klein, Hedrick et al. 2004). This is one of two classical viewpoints of space: space consists of spatial elements and the properties of space are the result of the properties of the spatial elements. Most of the properties of spatial elements depend on the metric defined for the space, they are metric properties. From the position of points in space most metric properties of a

geometric figure can be deduced; positions in space are expressed in computers with approximations.

Produce values to describe points in

The first chapter in this part separates different types of geometries; it concentrates on what remains invariant under transformation. Each 'geometry' defined by properties invariant under a transformation, defines one of the different ways we conceive space and time. The second chapter concentrates on observation of duration and time points. The third chapter then introduces coordinates to represent points in space in a computerized information system. The last chapter of the part covers transformations of coordinate space. The part reaches some unification of different aspects of transformations of space and time into a single formalism.

Chapter 7 CONTINUITY: THE MODEL OF GEOGRAPHIC SPACE AND TIME

Applications of GIS use different models of geographic space and time. Consider how a tax assessor appraises a property: he considers the area and the frontage of the parcel and weights the value by the distance to the city center (Figure 65). This uses three different *'spaces'*: an areal and a linear space in which the property is evaluated and a gravity model of space with a decay from a center for valuation(Abler, Adams et al. 1971).

A GIS must be capable of integrating these different conceptual models in a single formal system. The chapter gives a partial answer to one of the fundamental question of GIScience, namely "What is special about space?" (Egenhofer 1993)and justify what a theory of GIS must achieve: integration of different aspects of space and a uniform treatment of different representations of space. Considering the connection between transformations and properties which remain invariant (unchanged) by them gives us a handle to classify different parts of geometry: e.g., affine, projective geometry, or topology.

The chapter differentiates types of geometry by groups of transformations and what properties they leave invariant. The questions, which should be answered here for each different concept of space, are:

- Why are there multiple representations?
- What are the transformations?
- What are the invariants?
- What are the operations necessary?

A discussion of time follows in the next chapter.

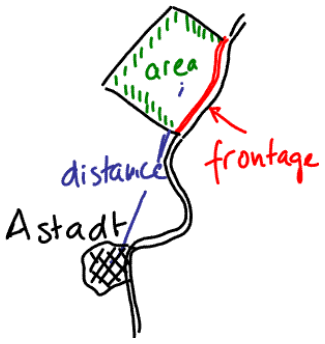


Figure 65: Geometric elements used to assess value of a property

1. DIFFERENT GEOMETRIES

The discovery of other than the "ordinary" geometry described by Euclid was an "intellectual revolution" that changed the way the world was seen(Blumenthal 1986). Kant had postulated that Euclidean geometry was inherent in nature ("god given") and his opinion weighted so much, that the eminent mathematician C.F. Gauss was not willing to publish his discovery of other than Euclidean geometries, about which he wrote in a letter 1824. The

original publications are by Nikolai Lobachewsky in 1829 and Johann Boylyai 1832 a few years later.

Non-Euclidean geometries were discovered in an effort to proof the independence of Euclid's axiom about parallel lines. Euclid stated five axioms for classical geometry constructed with ruler and compass: all observed properties of geometric figures follow from these axioms(Heath 1981). The properties of space and geometry seem to be captured in these axioms and not limited to measurements and numbers! It may be useful to reproduce them here:

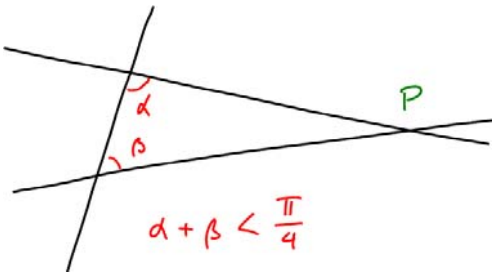


Figure 66: Euclid's fifth axiom: P is on the side where the angles are less then two right angles

"Let the following be postulated:

I. To draw a straight line from any point to any point.

II. To produce a finite straight line continuously in a straight line.

III. To describe a circle with any center and distance.

IV. That all right angles are equal to one another

V. That, if a straight line falling on two straight lines makes the interior angles on the same side less than two right angles, the two straight lines, if produced indefinitely, meet on that side on which the angles are less than two right angles." (Figure 66) (Blumenthal 1986, 2)

Lobachevsky demonstrated that a logical system with a negation of the fifth axiom is consistent. He expected a contradiction which would have shown the dependence of the fifth axioms from the other four axioms. The new consistent system of axioms produced a new logical system for geometry, a geometry with axioms different from the ones given by Euclid. One of these non-Euclidean geometries, namely projective geometry, in which all lines intersect, will be used in chapter 19 to find a closed formula to compute line intersections and avoid the ordinary Euclidean computations that require special treatment for parallel lines.

The discovery of non-Euclidean geometries led later Einstein to the formulation of relativity theory. Geography deals with objects and spaces that are limited to the earth and we are not concerned with large distances and, correspondingly, very high velocities. For the purposes of geography, not however for geodesy, Euclidean geometry is sufficient and relativistic effects are not relevant. We can ignore the Lorentz-transformations

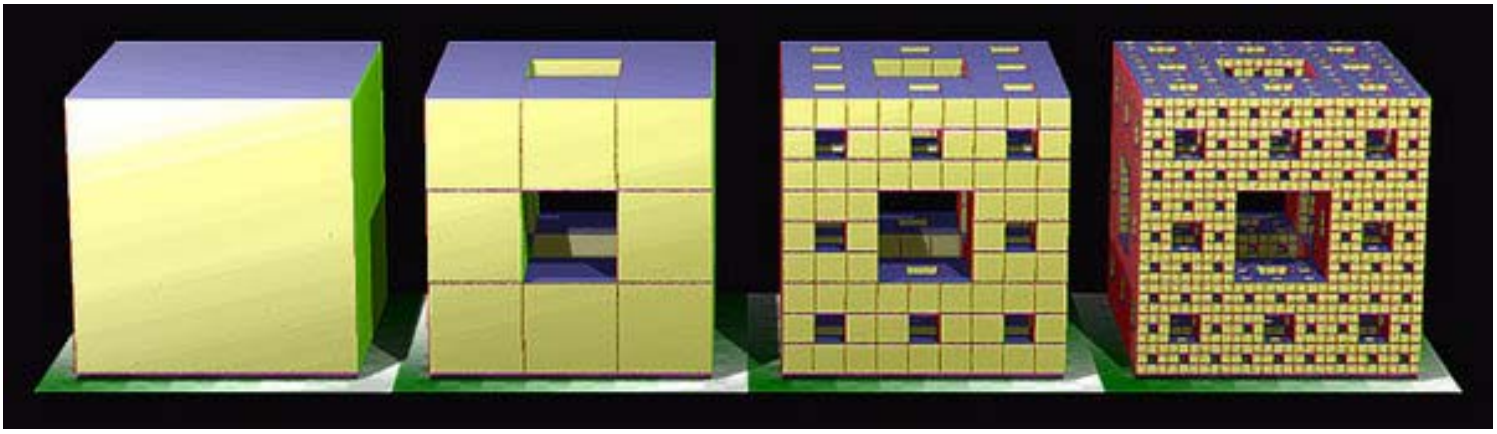


Figure 68: An example of a volume not physically realizable: a Menger sponge (from Wikipedia: Fractals)

(Figure 67), which reduce for all movements that are slow in comparison with the speed of light to the ordinary Galilean-Transformations. However, they are a prominent example for a special concept of geometry, which includes relative movement; the Lorentz-transformations also demonstrate how a general theory reduces to a simpler theory in ordinary cases.

Galilei-Transformation for moving metric syst.

$$x' = x - vt$$
$$t' = t$$

Lorentz-Transformation

$$x' = k\,(x - vt) \qquad \text{where} \quad k = 1/\sqrt{1-\left(\frac{v}{c}\right)^2}$$
$$t' = k\,\left(t - \frac{vx}{c^2}\right) \qquad c = \text{speed of light}$$

Figure 67: Transformations of space

The power of abstract geometry, allows many kinds of geometric objects that cannot exist in reality (Blumenthal and Menger 1970; Galton 2000, 502)(Figure 68). The art of modeling geometry in GIS is to find subsets of geometries that cover the cases that are possible with physical objects and correspond to our experiential abilities limited by our senses. Different special cases were studied individually but a GIS requires a combination of them.

2. Different Models for Different Applications

Space and time is fundamental for biological life—all people and animals are physical bodies that occupy space and move around in space(Couclelis and Gale 1986). Space is also fundamental for human cognition. Our daily experience with space and in space shapes our theoretical understanding of space and time(Lakoff and Johnson 1999). This understanding is formalized as geometry (Lakoff and Núnez 2000) and needs implementation in a computer system that deals with spatial information.

Experience with space varies depending on the goals we pursue: we may walk in space moving from one point to another, we may till the land for agriculture, and we may construct dwellings. But space is also involved when we draw a picture on paper, when we arrange the tools on our workbench, etc.(Couclelis 1992). Couclelis and Gale have discussed the different aspects of space and time: the concept of space used in mechanics, where motion can be reversed, is different from the one used in biology, where heat is dissipated and change cannot be reversed(Couclelis and Gale 1986). In human cognition, space is differentiated by the size of the space and how it is apprehended. Space with 2-or 3-dimension and time can be merged in a 3- or 4-dimensional physical or continuum, but human experience with these dimensions is different: time cannot replace space and even in space align, the vertical direction is more salient than front-back or left-right. This motivates different conceptualizations, which are each optimized for some applications.

- "Figurative space is … smaller than the body, its properties may be … perceived from one place without … locomotion".
- "Vista space is larger than the body and … can be visually apprehended from a single place without … locomotion".
- "Environmental space is larger then the body and surrounds it." It cannot be apprehended directly without considerable locomotion and requires "the integration of information over … time".
- "Geographical space is much larger than the body and cannot be apprehended directly through locomotion; ..., it must be learned via symbolic representations."(Montello 1993, 315).

Differentiate 4 kinds of space:
- figurative,
- vista,
- environment, and
- geographic.

For physical analysis, motion in space assumes continuous time and continuous space and the motion itself is continuous. Human conscious thinking about motion and change reduces these continua to discrete entities, which are represented in the cognitive system (Figure 69),(Kuipers 1994; Galton 2000, 321).

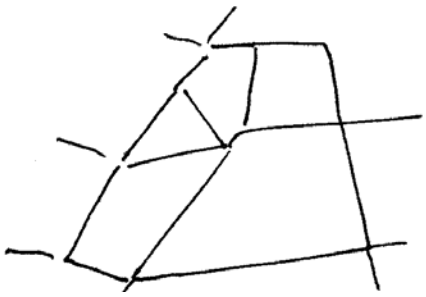


Figure 69: Graph representing the Street Network around TU Wien

The representation of continuous time and space in a discrete form is fundamental to human reasoning with space—and it seems that each approach captures some features for one class of activities leaving out others that are less for this application. The ways people treat different spatial experiences motivate the different discretizations used for continuous space and time. Different discretizations are essentially different theories of space, different geometries so to speak: the geometry of graphs is motivated by the network of streets (Figure 69), ordinary Euclidean geometry by the movement of rigid objects in space (Figure 70).

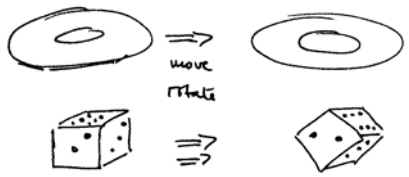


Figure 70: Euclidean geometry of solid objects

3. SPACE ALLOWS AN UNLIMITED AMOUNT OF DETAIL

Space, like time, can be observed at different levels of detail. We select the appropriate level for the task at hand and observe more precisely, when more detail is necessary(Timpf, Volta et al. 1992; Timpf and Frank 1997; Timpf 1998). There is always more detail possible: from a map 1:1 Mio, we can go to a map 1:200,000 and then to 1:50,000, etc. But this does not end with maps 1:50; maps of 1:1 are possible (Caroll 1893; Borges 1997) and even a map 10:1 can be drawn—there is detail in space to be shown, even if we cannot see it with our eyes directly.

The same applies to time: finer resolution is always possible. Actions are composed of smaller and smaller acts; often we are not aware of the finer level of temporal resolution because the activities at this level of detail are not visible to us and are of no interest in normal circumstances. Molecules move in an arbitrary movement, the speed of which is proportional to the temperature of a substance—we are satisfied with the temperature reading and are not interested in this Brownian motion.

3.1 MAP SCALE AND LEVEL OF DETAIL

Scale: ratio between distance on map and distance in reality.

Map scale is defined as a numerical factor, obtained from dividing the distance in a map by the corresponding distance in reality. This concept of scale is of little use in computer representations, where points are represented by their (real world) coordinates; scale is necessary when existing maps are digitized or data is visualized as maps.

The map scale implies also how much detail from reality is selected and represented on the map. The concept *Level of Detail* describes this better. The physical world in space and time can

be observed at (practically) unlimited level of detail. There are atomistic limits, but these are not relevant for a discussion of geography—geographic objects are many orders of magnitude larger than the smallest particle that we consider as undividable (atomic). For each representation a level of detail must be selected. Tobler has pointed out that we can detect objects which are n meters large, if the scale of the map is 1/n*1000; for example, in a map 1:100,000, objects of 100 m size can be detected(Tobler 198?).

3.2 Self-similarity and fractal dimension

Continuity avails more detail as we increase the resolution. This leads to a question for measurement: at what level of detail is the correct observation? Richardson (quoted by Mandelbrot 1977) has observed that measuring the length of a coastline depends on the level of detail with which one measures. If we measure the length of a line with a compass set to a fixed length and count how often this unit is in the line (Figure 70) the result varies with the size of the unit distance. If you repeat the experiment with a smaller unit distance, the total length of the line becomes longer (Figure 71). The length of the line depends on the level of detail of the representation considered(Buttenfield 1984; Buttenfield 1989).

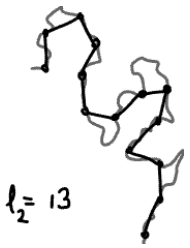


Figure 72: Coastline measured with yardstick of 1 unit

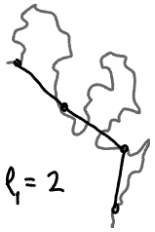
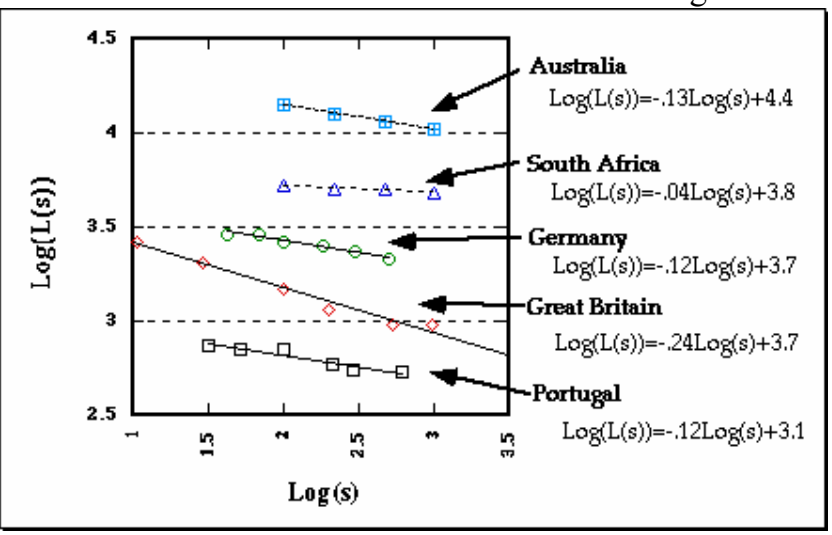


Figure 71: A coastline measured with yardstick of 2 units

The increase in length is a function of the reduction of the unit with which one measures. Following Mandelbrot, the ratio *log length / log unit is* called the fractal dimension of a line; a straight line has dimension 1; its length does not increase if we use a smaller yardstick!

A line with fractal dimension 2 fills all of *2-dimensional* space. Ordinary curved lines have a fractal dimension between 1 and 2. Figure 73 shows a construction of a fractal line with

Figure 73: Fractal dimension (from http://www.vanderbilt.edu/AnS/psychology/cogsci/chaos/workshop/Fractals.html)



fractal dimension 4/3. Mandelbrot has pointed out that fractal lines are self similar, each part has the same form as the whole(Mandelbrot 1977), which, to a certain degree, is true also for geographic phenomena, e.g. coast lines. This relation between length of a line and resolution with which one measures applies not only to the length, but to other observations as well(Batty and Longley 1994; Quattrochi and Goodchild 1997). Openshaw has studied extensively the 'movable areal unit problem' (MAUP): what is the correct resolution to study for example unemployment rates: the block, the town, the county, or a whole state? Different results obtain! (Openshaw and Alvanides 2001)

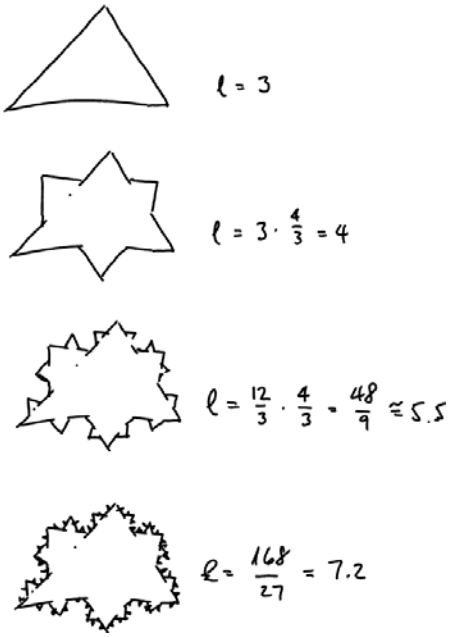


Figure 74: Koch's snowflake. A line with fractal dimension 4/3

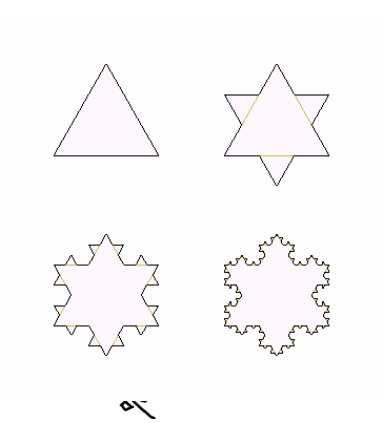


Figure 75: A town at different levels of detail

4. MULTIPLE REPRESENTATION

The infinite amount of detail potentially available requires multiple representations of the same reality, at different scales and with different intentions(Buttenfield and Delotto 1989; Günther 1989; Frank and Timpf 1994; Timpf and Devogele 1997). A town can be shown as a point, an area, a grid of major roads, a collection of buildings, etc. (Figure 75). Some of these representations use different types of geometries: for example a road can be seen as a volume of building materials, an area (as a street parcel), or as a street line connecting two intersections.

The description of the methods to treat aspects of geometry is only the conceptual foundation(Timpf 1998). For real systems, we must be able to link different representations together and use reasoning across representations(Timpf, Volta et al. 1992).

5. SPACE AND TIME ALLOWS MANY RELATIONS

Between objects in space and time many different relations can exist. We can consider topological relations, like "Jamaica is an island in the Caribbean" or the distance between Vienna and Salzburg and compare it with other distances between cities. There are nearly infinite many relations between objects in space and it is impossible to represent them all explicitly. A GIS with 10000 named places could have 50 million distance relations between them. The often seen tables for distances between villages work only for a small island like Elba.

It is to identify those properties from which other properties can be derived. Only the first need be explicitly stored, the others

Distanze fra i capoluoghi dei comuni all'Isola d'Elba:

Portoferraio						
16,4	**Capoliveri**					
14,6	5,4	**Porto Azzurro**				
24,9	15,7	10,3	**Rio nell'Elba**			
26,6	17,4	12,0	3,3	**Rio Marina**		
17,8	20,5	18,7	29,0	30,7	**Marina di Campo**	
19,8	30,2	28,4	38,7	40,4	12,8	**Marciana Marina**
27,2	36,1	34,6	44,9	46,6	18,9	7,4 **Marciana**

I valori riportati nella tabella corrispondono alla distanza minima tra località su strada asfaltata.

are computed. For example, a GIS stores the location of places as coordinate pairs and computes the distances.

Relations that are not changing are most useful to remember. We call such relations invariant. For example, the length and width of a taxi cab remains the same, while the location of the taxi is changing. Concentrating on properties that remain valid despite other changes reduces the need for constant updating of the relation or property. It is economical to identify the key properties from which others can be deduced.

6. Differentiation of Geometries by What They Leave Invariant

'Invariance under a group of transformation' can be used to divide geometry in logically connected subfields as suggested by Felix Klein in the "Erlanger program" (Klein 1872) and links directly with our experience: The use of objects is determined by what we can do with them—for example move them in space—and what properties they maintain. These are the properties that

Figure 76: Different solid objects

Figure 77: Different forms, but the same topology

Figure 78: A balloon changes its form

determine the preferred spatial concept for this application; we may say, they determine the "geometry of this application".

The prototypical geometric objects are small, movable, rigid bodies (Figure 76), which preserve their geometry even when moved in figurative space. Continuity of space is preserved even in objects that are not rigid: continuity is preserved in garments, which do not have a definite form, but can be folded to be put in the wardrobe and then put on, hung an a hook, etc.(Figure 77). Continuity is preserved in objects that are even more flexible, e.g., rubber sheets and balloons, which can be deformed in many ways, but always preserve continuity (Figure 78).

Klein has proposed (1872) to study as *geometry* properties that remain invariant under a group of transformation. This abstract viewpoint captures practical aspects of objects. Rigid objects like a sword, a cup, or the triangles and rulers used for geometric constructions (Figure 76) 'work' only because they preserve a set of properties—a sword made from rubber does not work in the intended way, nor does a ruler. Garments made from rigid materials like tin foil were an interesting idea by Paco Rabanne for Haute Couture, but definitely not what we want to use everyday! It is essential for garments to be flexible, but preserve continuity (Figure 77). The same for a balloon—if it is punctured and bursts, it is not a balloon anymore.

Klein required that the transformations considered form a group (see chapter 5), meaning that there must be a unit transformation, an inverse to each transformation and transformations can be composed. These group properties are essential for the concept of a spatial transformation—if a transformation cannot be undone by its inverse or if there is no option of doing nothing, then it seems not to be a geometry. These requirements capture the properties of abstract physical space, not the living space of animals and plants, where movements cannot be completely undone as energy is dissipated(Couclelis and Gale 1986).

7. Different Types of Geometry Defined by Group of Transformations

Modern mathematics works towards unification: different theories should be brought into a single context, connected to work together and to be constructed on a common foundation. This requires that common concepts are 'factored out'. For

example, what are the common properties of different geometries, what is the essence of geometry(Blumenthal and Menger 1970). This is the same question as Egenhofer posed in "what is special about spatial?" (Egenhofer). A more geographic but similar viewpoint is found in Abler, Adam and Gould's work "Spatial Organization" (Abler, Adams et al.).

Factorization and its use in ordinary arithmetic:
*35 + 25 + 15 = 5 * 7 + 5 * 5 + 5 * 3 = 5 * (7 + 5 + 3) = 5 * 15 = 75*

A geometry is a group of mappings *M* of a space *S* onto itself, where the geometry studies properties of a figure (a subset of *S*) that are invariant under each transformation of the Group *M(Blumenthal and Menger 1970, 25-26)*. This definition is general and includes not only all what is usually studied under the notion geometry, but also, for example, relativity theory, which is the theory of the invariants of a 4-dimensional continuum (Minkowski's world) with respect to a given group of collineations (the Lorentz group).

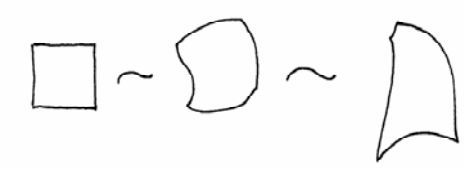


Figure 79: Object deformed

The geometrical essence of the definition of Klein is the equivalence of transformed figures and the properties of these equivalent figures. For example the three shapes in Figure 79 are equivalent under topological transformations—topology deals with the properties that they have in common and which are invariant under these transformations.

Blumenthal (Blumenthal and Menger 1970, 27) defines a geometry as

> A geometry *G* over a set Σ is a system $\{\Sigma, E\}$, where *E* denotes an equivalence relation defined in the set of all subsets (figures) of Σ. The geometry $\{\Sigma, E\}$ studies those properties of a figure *F* that the figure has in common with all figures equivalent to *F*; these are the invariant properties.

Geometry is the study of automorphism groups.

8. TRANSFORMATIONS USEFUL FOR DIFFERENTIATION OF GEOMETRIES

8.1 RIGID BODY MOTION

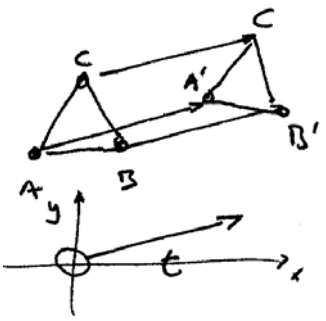


Figure 80: Translation

These transformations describe the movement of rigid objects. They have various names: they are sometimes called Euclidean (Hartley and Zisserman 2000) or congruence transformations, because figures remain congruent. They are transformations of space, which preserve distances between the parts of the objects—this is the essence of rigidity. Rigid body motions can be separated in translations and rotations.

8.1.1 Translation

Translations form a group of transformations (Figure 80): every translation has an inverse and doing nothing is a zero translation. Translation leaves distances invariant.

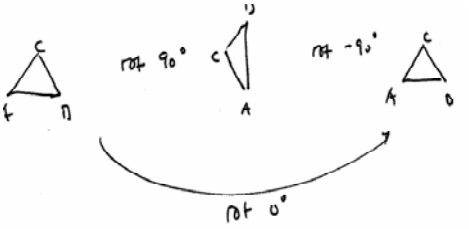


Figure 81: Rotation

8.1.2 Rotation

Rotation (Figure 81) forms a group, with the rotation with angle *0* as zero and the inverse rotation is the rotation with the reversed angle. Rotation leave distances invariant.

8.1.3 Congruence relations preserve angles

Translations and rotations leave distances unchanged and necessarily also angles. We speak of metric properties when discussing the preservation of distances and angles. Translation leaves invariant azimuth, which is the angle between a line connecting two points and one of the base vectors of the space. Rotation does not preserve azimuth (Figure 83: Azimuth (positive turning)).

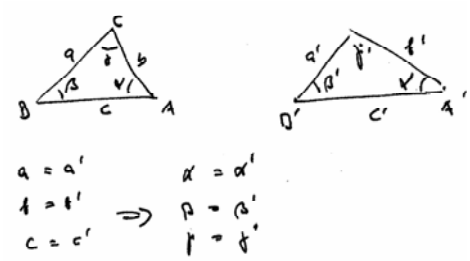


Figure 82: Congruence Transformation

8.2 ISOMETRY

Isometries are all transformations which leave distances and angles invariant. They are the rigid body motions and the reflections. Reflections leave distances invariant, but reverse the direction of angles.

Figure 83: Azimuth (positive turning)

8.3 SCALING

Scaling forms a group of transformations, with the unit transformation is scaling with the value *1* and the inverse scaling is scaling with the (multiplicative) inverse scale. This leaves invariant **azimuths** and **angles**, but not distances. Areas are multiplied with the square of the scale factor.

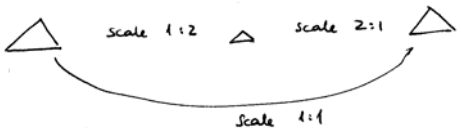


Figure 84: Scaling

8.4 SIMILARITY TRANSFORMATIONS

Similarity transformations leave the proportions of metric properties the same ; they consist of translations, rotations, and scale changes. This is ordinary Euclidean geometry, where circles remain circles.

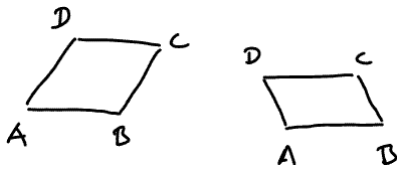


Figure 85: Affine Transformation

8.5 AFFINE TRANSFORMATIONS

A generalization of linear transformations leads to affine transformations (Figure 85), which include translation, rotation, and scale as special cases. They can result from parallel projection and transform parallel lines into parallel lines. They preserve the ratio of length of parallel line segments. Affine transformations can be seen as a composition of two different scales on orthogonal axis. The areas are multiplied by the product of the scale factors.

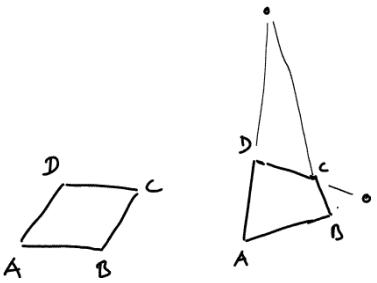


Figure 86: Perspective transformation

8.6 PROJECTIVE TRANSFORMATIONS

Projective transformations preserves collinearity (Figure 88) and the cross ratio (Figure 87)(Stolfi 1991, 123). They are a generalization of affine transformations.

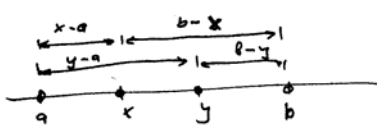


Figure 87: Cross ratio

$$\frac{(x:y)}{(a:b)} = \frac{x-a}{b-x} \Big/ \frac{y-a}{b-y}$$

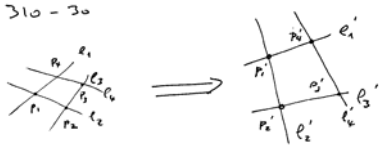


Figure 88: Preservation of collinearity

8.7 TOPOLOGICAL TRANSFORMATIONS

Continuous transformations are transformations that preserve neighborhoods. They do not allow puncturing, cutting, and gluing parts of objects together (Figure 89). Closing the legs of a pair of pants by sewing them shut is a practical joke and make the pants non-functional. Puncturing a hole in a cup renders it useless. Topological or *homoemorphic* transformations preserve these properties of objects, which are crucial for the function of an object.

Figure 89: Puncture a balloon, glue an envelope shut and cut off a coupon

Note: homomorphism and homeomorphism are two distinct concepts!

Topological transformations preserve neighborhoods.

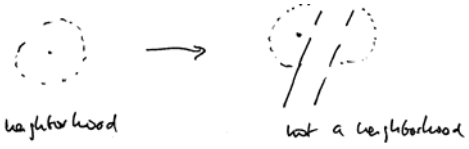


Figure 90: Neighborhood

9. MAP PROJECTIONS

Map projections are a special case of transformations, namely from the surface of a *3-dimensional* sphere to a *2-dimensional* plane. They leave incidences—points lying on a line remain on the line—but do not preserve angles, distances, and azimuths all at once. Geodesic lines are not always mapped to geodesic lines. Map projections do not form a group of transformations and do not define geometries in the sense of Klein's Erlanger Program. There are various optimizations to preserve *gestalt*, a concept that has not been expressed in mathematical terms. Many different map projections exist and they optimize different properties. The transformations are in general not linear. A systematic treatment of cartographic projections is not intended here(for more detail see Bugayevskiy and Snyder 1995).

10. SUMMARY

Many of the geometric constructions—especially the classical constructions of Euclidean geometry carried out through motions of rigid bodies (compass, ruler)—can be seen as translations, rotations, etc. and the problems of classical geometry formulated as transformations. These transformations have the properties of a group (*0*, inverse). Transformations can be composed. This makes a transformation based approach attractive, because composition of ordinary geometric constructions with compass and ruler are difficult to describe.

Geometry as transformation relates directly to the treatment of geometry in computers when point positions are represented with coordinates, and where transformations are expressed as linear transformations, that is, matrices with the ordinary operations of linear algebra (this is the topic of chapter 9 and 10). The relation between transformation and the properties they leave invariant is shown in Figure 91.

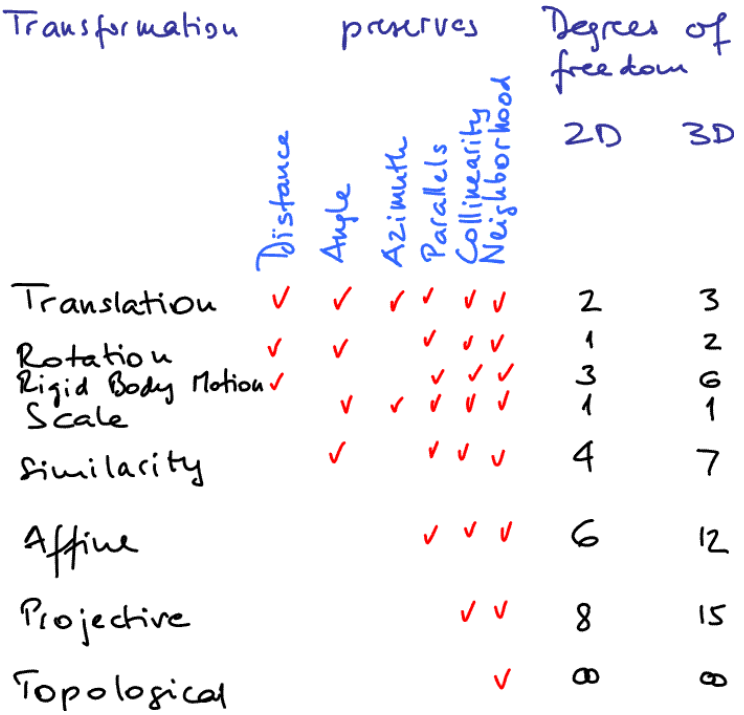


Figure 91: Different transformations and what they leave invariant

Space is continuous and contains infinite amount of detail; our conceptualization and representation picks out aspects that are relevant for some application; different applications pick different aspects. It is not possible to construct a single representation that suits all application areas, but we demand that data and operations from different application areas can be integrated. The question is to find a most general set of

operations that is applicable to many representations. In this chapter we have shown a classification; in the following chapters the differentiated parts of geometry are discussed individually, following this classification.

REVIEW QUESTIONS

- Where does the impossibility to represent continuous space practically show? Give example form real life and from information system.
- Explain the difference between physical and biological time.
- How was non-Euclidean geometry discovered?
- List three geometries and describe what transformations are permitted and what properties remain invariant.
- Why is it important that geometric transformations form a group?
- How many degrees of freedom has a projective transformation? How many a congruence transformation?
- Demonstrate that affine transformations form a group.

Chapter 8 TIME: DURATION AND TIME POINTS

Time is a fundamental dimension of reality: people and all other things exist and evolve in time. Support to manage temporal aspects is usually not included in GIS software but often demanded. For managing weather data, natural resources but also the cadastre and land registration, time is important(Al-Taha 1992).

A GIS that includes time needs a reference system to describe points in time, not only measured duration (chapter 6). This chapter deals with the conventional method of describing time points, for which we will also use the term 'instant' and how we convert time as duration (intervals) which we can measure, to time points, which we cannot measure. Converting and integrating time points observed in different reference frames pose a difficult problem.

Time observations and time points are covered before we discuss observations and points in space. The 1-dimensionality of time makes it easier to discuss time and shows the issues more clearly than when discussing space. There are more differences between time and space than just the difference between 1- and 3- dimensions. Time is fundamentally different from space: we can move freely in space, but not in time; time is ordered and there is a special point *'now'*, which is constantly changing. Position in space is observed to be able to return to this point – time points are observed only for synchronization, because we cannot return to a previous time ever. Day and night imposes on conventional time a regular 'natural' structure; space has—at best—an irregular structure, which we call geography.

1. INTRODUCTION

Time and space are the fundamental dimensions of the reality in which people live. Without time no change, but 'life is change', without time no life! Most GIS software today concentrates on the management of spatial snapshots and ignore time {Frank, 1998 #8250}. They show the geographic reality as an immutable, unchanging collection of facts. This may be a carry over from printed maps, which focus on objects, which remain unchanged for long periods of time. Cartography has only

limited methods to represent change (Tufte 1997)[PhD. diss with monmonier; possibly something by ncgia – babs?]; but with electronic media, there is no need to concentrate on the immutable part of reality. GIS could provide support for time and changing situations and many web services provide constantly changing maps of, e.g., the traffic situation in a region. Change, not a static situation, attracts attention; it is difficult *not* to watch something moving within one's visual field. Change in the socio-economic or the natural environment attracts the politician's attention and we should make any effort possible, to build GIS that can inform about change(Frank 1998).

Efforts to introduce time into computing, in particular into geographic data processing came only around 1988 with a thesis by Langran (Langran and Chrisman 1988; Langran 1989). The original NCGIA research plan (NCGIA 1989) included 'Time' as a special research focus and organized an initial meeting (Barrera, Frank et al. 1991) and later a specialist meeting(Egenhofer and Golledge 1994). In Europe a meeting was organized in the GISDATA series(Frank 1996). The Chorochronos project studied spatio-temporal databases(Frank 2003; Sellis and Koubarakis 2003). The book by Galton gives an AI perspective on time (Galton 2000) and Güting and Schneider give a database perspective restricted to moving objects [Güting and Schneider].

2. EXPERIENCED TIME

Time is experienced by humans in subjective, non-uniform ways: sometimes time flies like an arrow, sometimes waiting becomes unbearable and time progresses slowly. Do you remember how you were waiting for Christmas when you were a child? We will concentrate here on the objective view of time and assume an absolute time, which marches continuously and uniformly from the past through the *now* to the future (Figure 92).

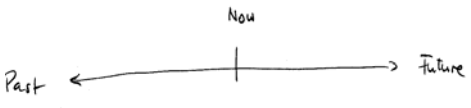


Figure 92: Time from Past to Future

Joke: Times flies like an arrow,
Fruit flies like bananas.

Figure 93: March towards the future

We customarily use two metaphors to conceptualize time: we (the *now*) is moving in time (Figure 93), or the time is rushing past us and we are fixed looking towards the future (Figure 94)—there is no difference between the two for the formal treatment. The third option: where the observer looks towards the past and the future is approaching unseen from the back (Figure 95) is customary for some American Indians; it

seems more fitting with the facts: we know the past and do not know the future.

We all are at the same point in time—the *now*, and can observe the world state at only this time. The *now* is the same for all of us and we can never return to it. This is different from space, where we can move freely and observe at arbitrary points, where different people have different perspectives.

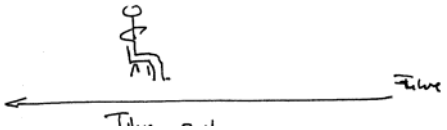


Figure 94: Time rushes past us

Figure 95: An Indian metaphor for time: it approaches us from behind

Time is a fundamental resource. Georg Franck has pointed out that a person's time is the only resource that is fundamentally scarce: every person has a lifetime—just one. An economic assessment of the resource 'personal time' leads to deep insights in how we manage attention and explains high payments to celebrities and the economy of the media in general(Franck 1998).

3. TOTALLY ORDERED MODEL OF TIME

Points in time are similar to points in space—they are dimensionless points, imbedded in the 1-dimensional time line. The time line is a single line, dense and continuous. (This model of time does not include the concept of a *now*). It is customary to represent time by *real numbers* and approximate them with floating point numbers in a computer. Using a dense and continuous time line allows to apply the apparatus of calculus to time, and later to space-time, which has demonstrated great merits in physics and engineering.

Dense—between any two instants there is another instant
Continuous—no gaps

Galton's model of time is totally ordered by a primitive relation before (<). Galton adds *unboundedness* to the axioms, stating that there is no first and last time point. Time can be either *dense*, meaning that between any two points is another point; or, alternatively, *discrete*, where there are immediately preceding and following time points, such that no other time points are in between. Either the dense or the discrete axiom gives together with the other axioms a consistent and syntactically complete set(Galton 2000).

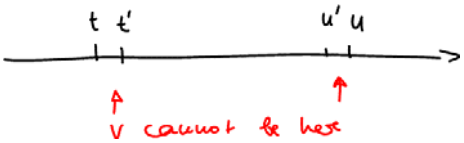


Figure 96: Axiom for discrete time

Totally Ordered Time <	
irreflexivity	*not (t < t)*
transitivity	*t < u and u < v => t < v*
linearity	*if t ≠ u then either t < u or u < t*
unboundedness	*For every t, exist u and v, such that u < t and t < v.*

dense	*For every t and u, t < u*	*exist v such that t < v and v < u*
or		
discrete	*For every t and u, t < u, there are instants t' and u' such that t < t' and u' < u, and for no instant v is it the case that either t < v and v < t' or u' < v and v < u.*	

4. BRANCHING TIME (TIME WITH PARTIAL ORDER)

The ordinary ontological commitment is assuming only one single world, which marches through time {Frank, 2001 #9957; Frank, 1999 #182; Frank, 2003 #9920}. Science Fiction is using other models of time, where parallel universes exist in their separate times(Asimov 1957; Adams 1979). These branching times are not only interesting to construct science fiction novels, but necessary to deal with plans for the future and to represent uncertainty about the past. Branching models of time are necessary for Game Theory (Neumann von and Morgenstern 1944) and can represent the uncertainty of events in the future(Galton 1987).

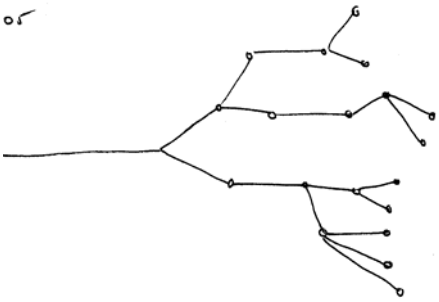


Figure 97: Different planned futures

Planning describes future states of the world. We make decisions between different courses of actions and reach then different states of the world. Alternatively, considering the current state of the world, we may hypothesize about different sequences of actions that have produced this state; this may be in a criminal story or describing geological processes that have produced the current shape of the world(Flewelling, Egenhofer et al. 1992).

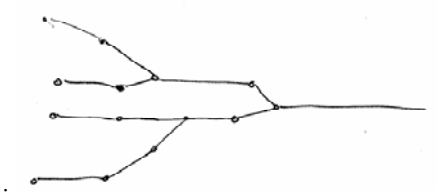


Figure 98: Hypothetical different pasts

4.1 UNCOORDINATED REPORTS OF EVENTS

Unrelated reports may give sequences of events, but not describe their relations precisely. The sequence of actions necessary to get to the office in the morning is the same for most of us: an alarm goes off, we get up, dress, have breakfast and then go to the office. If a day starts with both Dr. Navratil and me sleeping at 5 o'clock in the morning and later we meet at the office at 9 o'clock, then the events for each of us are totally ordered, but there is no order between events not in the same sequence; in chapter 16 we will introduce the notion of partial order to formalize this. We can not determine if I had breakfast before him or not (Figure 99).

Branching time is partially ordered

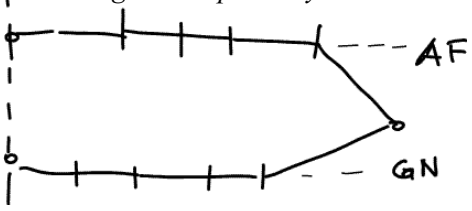


Figure 99: Two totally ordered sequences give a partially ordered sequence

4.2 CRITICAL PATH

Practically, models of branching time are for the determination of the critical path—that is the path that determines the minimum time necessary to achieve some future state. In a Critical Path (or Program Evaluation and Review Technique PERT) diagram, the arrows represent activities that have a minimal and maximal duration and the nodes are milestones when a defined state is reached (e.g. the excavation for a building is completed). It is then possible to calculate the earliest and the latest time possible a state is achieved. The path with the longest minimal time determines when a state can be achieved the earliest and is called the *critical path* to this event; only speeding up actions on the critical path leads to an earlier achievement.

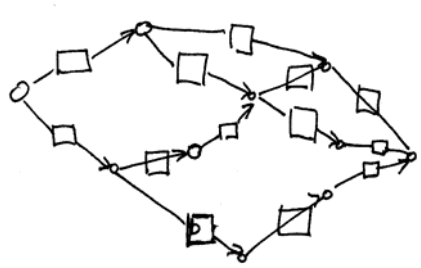


Figure 100: Critical Path Diagram

4.3 GAME THEORY

Game theory considers in its simplest case a special case of branching time: in a two person game, the two adversaries have each one decision to make and the outcome of the game (i.e., the future state of the world) depends on the two decisions.

Game theory evaluates the future state from the perspective of each player and gives rules, what action a rational player will select, and thus what you have to expect from a rational opponent(Neumann von and Morgenstern 1944). Game theory has found many applications in economy (Davis 1983) and even law (Baird, Gertner et al. 1994)

4.4 PROBABILITY OF FUTURE STATES

Sometimes the transition from a current state to a future state are taken with a known probability, diagrams show the combined probability to reach different future state. They are useful to assess the likelihood of serious accidents that result from the unlikely combination of small errors—for example in the management of nuclear plants.

5. DURATION (TIME LENGTH)

Time is measured as duration—even if it appears that we determine duration as the difference between two time points. Duration is a measurement, expressed in seconds or multiple of seconds. It is a ratio type (see chapter 6.9xx). For duration the same operations than to other measurements apply: addition, subtraction, multiplication and division with a scalar and ratio,

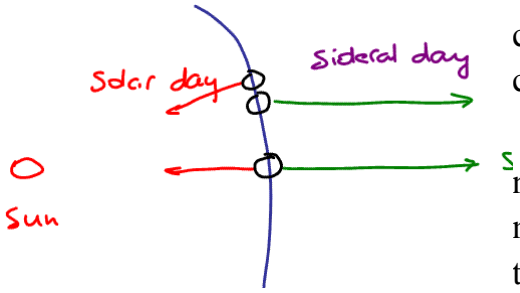


Figure 101: Tropical and sidereal day

comparing two durations (see algebra for measurements in chapter 6xx).

The SI unit is the second—which is defined today as a number of oscillations of a well defined atomic state and multiples of seconds: minutes, hours and days, based on traditional Babylonian divisions in 60 and 12. Week is the longest commonly used time unit, which has a fixed length. Neither month nor year have always the same length, but are commonly used as if they were of fixed length!

For scientific purposes, especially astronomy, other definitions of day and year are used, based on the rotation of the earth and the movement of the earth around the sun (Figure 101). These exact definitions of the length of day and year seem not to be used in GIS.

A mean tropical year has 365.2422 SI days

A sidereal day has 23 hours 56 m and 4 seconds

6. Instants and Intervals

One can take instants as primitives and construct intervals from them (Galton 1987) or to take intervals as primitives and construct instants from them(Allen and Hayes 1985); we follow here Galton's approach, which translates later more directly to an implementation.

Instants have no duration, they are points in time.

The technology for measuring time is measuring time intervals, but synchronization between clocks is so advanced that the illusion of measuring time points directly is achieved. Accurate radio signals giving time in the absolute frame of UTC (Coordinated Universal Time), which is the mean time of the Greenwich Astronomical Observatory (in London, UK) and used for all civilian applications.

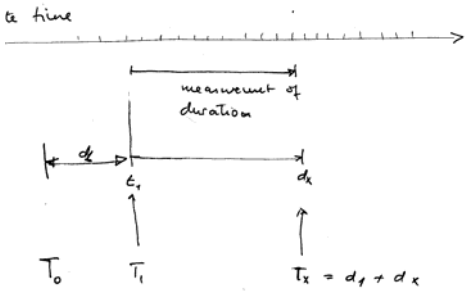


Figure 102: Measurement of duration gives absolute time

7. Granularity of Time Measurements

Time, like space, can be investigated at different levels of resolution. Depending on the task we are interested in, time is measured in years, days, seconds, milliseconds, etc. The precision with which we measure time varies and is often fixed for application areas: in commercial banking, duration is measured in days and all time points within a day are considered as happening at the same time; banking is a cyclic operation, with a cycle per day(Frank 1998). In a traditional world, where nights of silence and rest separate days of activity, this makes perfect sense—but in today's global economy, where stock is

traded around the clock in one or the other stock exchange around the globe, such conventions lose their force.

Customary Time Intervals are defined as half-open; they include the start point, but not the ending point.

For administration, all events during a day or a year are considered concurrent, whereas events, only seconds apart but in different years, are treated differently. Usually days go from midnight 00:00 to 23:59:59, and similarly for month, year etc. All the customary intervals—seen as container—do include the 0 moment, but not the ending moment.

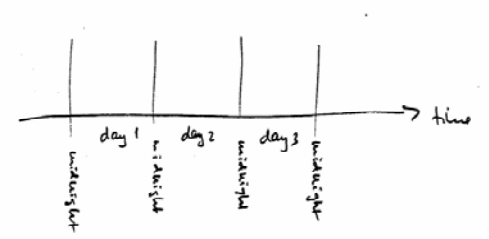


Figure 103: Granularity of Time

This is different from measurements of limited precision in space, where there is no dominant subdivision against which measurements are taken. The subdivision of space is irregular—e.g., the political subdivision (Figure 104)—and provides a frame for imprecise indication of location(Bittner 1999; Bittner and Smith 2003; Bittner and Smith 2003; Bittner and Smith 2003 (draft)); but this is not treated as measurement. Rome is in Italy is an indication of location comparable to x was born Feb 10, 1982.

Figure 104: Irregular granularity of space

8. ORIGIN OF THE TIME LINE

To determine time points an origin must be selected and time points are determined by measuring the duration of the interval from the origin to the desired point. Astronomical observations are used to establish new, derived points of fixed and known distance from the selected origin.

The origin of time systems for our western calendar, the supposed year of the birth of Christ is used and years are measured from AD 1 following. The conventional system assumes a year 1 BC, immediately followed by a year AD 1 (there is no year 0). The creation of the earth is the origin for the civil Hebrew calendar, conventionally at 3 760 BC, and the escape from Mecca of the Prophet Muhammad, 622 AC is used as the year 1 in the Arabic and the Persian calendar. The Hebrew calendar is lunisolar, the Arabic is lunar and the Persian is solar and in consequence the number to subtract from a western year varies over time.

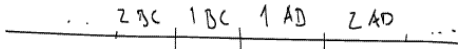


Figure 106: No year 0

Around 2006 the difference between western and
- Islamic year is 579
- Hebrew year is 3760
- Persian year is 721

The length of the year is not an even number of days but 365.2422 days and the difference is absorbed in a leap day in February every 4th year, but not when the century is dividable in 4. This current calendar is the result of the reform by Pope Gregory XIII in 1582; this reform was not accepted by the

Orthodox Church and became effective in Russia only with the Revolution.

```
leapYear y = ((mod y 4 == 0) && (mod y 100 ≠ 0)) ||
     (mod y 1000 == 0)
```

Conversion of historic dates and time is surprisingly complicated! The year in medieval time started with Easter, whereas the year today starts January first. Time within a day is now measured from midnight, but a few centuries ago, each town had its own convention. For example, Venetian time in the 18th century counted local hours from sunset onwards, which varies during the year.

Fast and regular transportation with railroads made it necessary to abolish a different local time for each town and to establish time zones. Within a time zone the local time of the central meridian is the uniform time for the whole zone; the zones are extended to what would be geometrically necessary to keep areas of intense commercial connections in the same time zone. A time point measured must be marked with the time zone in which it was made to allow comparison with other time observations in other zones.

The time indicated by a sun dial differs up to a 15 minutes from a uniform, mean local time!

Daytimes are influenced by the so-called Daylight Saving Time (in Europe called ‘summer time’), which is a change in the time of a zone to 1 hour earlier than the normal time. It is believed to reduce the energy consumption by shifting human activities further to the morning. The switch between normal zone time ("standard" time) and Daylight Saving Time is not everywhere at the same date; adding complexity to the conventional time measuring system.

8.1 Integration of timed measurements from different time zones

GIS integrate data collected at different locations and with respect to different time systems; modern data collection in geodesy is using UTC routinely, but other data collection efforts may use local time. For example, the collection of benchmark data for water levels at the Danube River uses 2 time zones and different Daylight Saving Time schemes may apply.

Difficult is to test if two events can coincide or not, if they happen in different time zones and the descriptions are not precise. For example, can an event that happened during May 25 in Orono, ME and an event happened during May 24 in Vienna,

Austria, coincide. Convert both days in intervals in GMT and then identify the common interval (Figure 107).

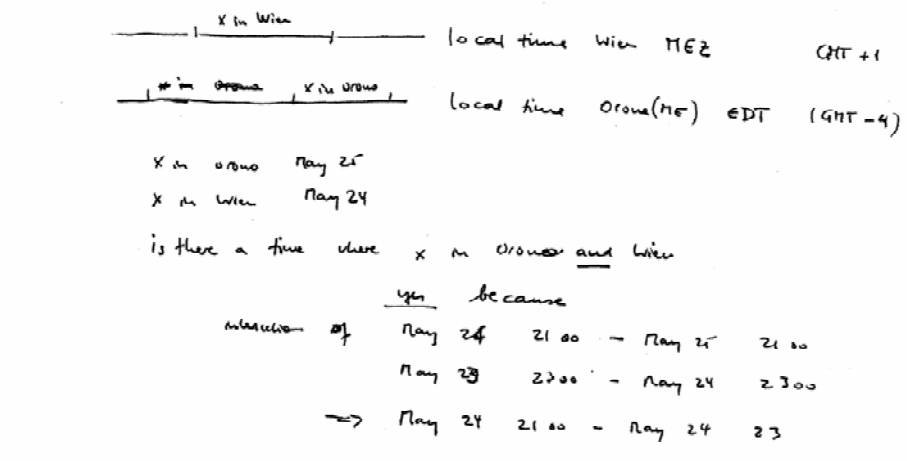


Figure 107: Time points expressed in different time zones

9. Conversion of Dates and Arithmetic Operations with Dates

The conversion of dates must consider what the origins are and that the numbering of days starts with 1, not with 0 as in other measurement lines (Figure 108). The most general way to compute with dates is to count them from a fixed origin. Convenient are dates like Jan 1, 1900, but any other date would be as good. It is desirable that no dates before this origin are ever used.

Once conversion from the customary date descriptions to a number of days since an origin has been accomplished, the computation with dates becomes simple additions or subtractions. How to add 17 days to Feb 24th? The result depends on the year—in leap years the result is March 12 and in other years it is March 13.

toDays (24 Feb, nonLeapYear) = 55
fromDays (55 + 17, nonLeapYear) = March 13
using formDays (toDays(x, nonLeapYear), nonLeapYear) = x
or fromDaysNL . toDaysNL = id

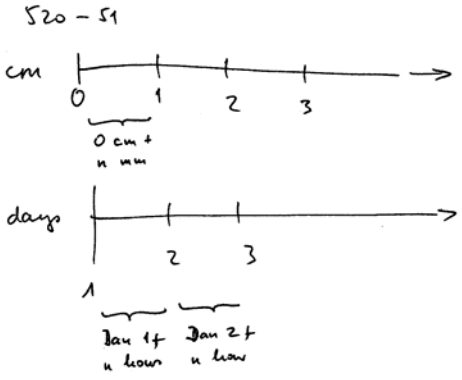


Figure 108: Counting of days is different from length

Time points expressed as days or hours have granularity. They are converted to intervals for computation. The length of the interval between two dates d_1 and d_2 expressed in days is not $d_2 - d_1$ but the granularity g (=1 day) must be added: $d_2 - d_1 + g$*(Tansel, Clifford et al. 1993)*, or subtracted - depending if we want to obtain the longest or the shortest duration between the two dates (Figure 109). For banking, if you pay interest, the longest interval is used, if you receive interest, the shortest is used!

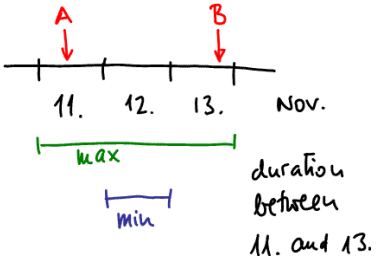


Figure 109: Duration between two dates

10. SUMMARY

To define the operations on time we need:

- A value of type time, which represents measurement of time intervals with the regular arithmetic operations for measurements.
- A type to represent time points, measured from a conventional origin with operations to convert conventional days into this type and from this type. This time with fixed origin converts time points in duration from the origin and makes the arithmetic operations for measurements applicable.

The UTC time would fulfill these requirements and makes – theoretically – calculation with time and dates simple. Various conventions on origin of a time scale and its subdivision in years and month, the granularity which applies to certain types of events etc. make integration of times on a global scale difficult and the problems become even larger if somebody constructs a GIS for historic times!

REVIEW QUESTIONS

- Why are time intervals defined as half-open?
- What is the meaning of a negative data (minus July 7)?
- How many years between 10 BC and 10 AD?
- What are intervals, what time points?
- What is the difference between point 3:15 and duration (3h 15m)?
- Determine the date 45 days after Jan 15th?
- How long lasted an event, starting Aug 1 and ending Aug 12? What is the maximal and what the minimal duration?

Chapter 9 SPACE: METRIC OPERATIONS FOR POINTS AND VECTOR ALGEBRA

Geographic Reality:
properties are observable for each point in 3d space and time:
F (x, y, z, t) = a

In this chapter the familiar concept of coordinates that describe points is introduced. A coordinate space is an intuitive model for space. Goodchild used it as a foundation for his definition of "geographic reality"(Frank 1990; Goodchild 1990). It is an example of the *application of a functor*. Scalars, e.g. real numbers, are sufficient to describe points on a line – for example time points—but are not sufficient for points in *2d* space. Points and operations with points form an algebra that captures essential properties of our concepts of space, namely transformations that form group and leave distance invariant(Klein 1872; Blumenthal and Menger 1970). This gives a treatment that is independent of the dimension, but the discussion and the examples here are for didactic reasons in terms of *2-dimensional* space.

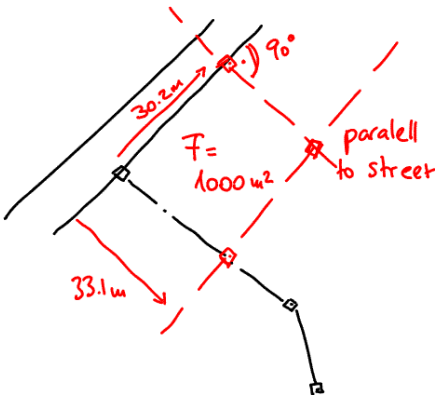


Figure 110: Example of a construction of a new parcel using COGO

Reminder:
only distances and angles can be measured, not coordinates (not even with GPS)!

The introduction of vectors here follows the construction of a module from a group and a ring as described in any algebra text book (Gill 1976; Mac Lane and Birkhoff 1991; Reinhardt and Soeder 1991). Vector algebra and vector space are abstract concepts that are not dependent on a coordinate frame, only their analytic treatment is. The algebra of vectors as it represents our manipulation of the geometry of rigid objects is mapped to computational operations on coordinates; the geometric properties, for example distance, are preserved across this transformation.

Vector algebra is used in a GIS in many ways, most of them not directly visible to the user. It is used when constructing a parcel from bearings and distances measured between the corner points (Figure 110); land surveyors have used such operations in computer programs called COGO (Coordinate Geometry) even before GIS(Miller 1963; DEC 1974).

1. Geometry on a Computer?

The Greeks did geometry with ruler (straight edge) and compass. Lines and circles—or rather approximations for these ideal figures—were drawn in sand. The reasoning however was not about the approximate figures, but the pure concepts of point and line, the so called Platonic ideals.

Descartes described in the 17th century a mapping from geometric construction to computations: analytical geometry was invented! Mapping real space to the coordinate space—the domain of pairs of real values—allows computational operations with real numbers that correspond to the geometric operations with ruler and compass in the plane. For example: given two points, the point in the middle can be computed (Figure 111).

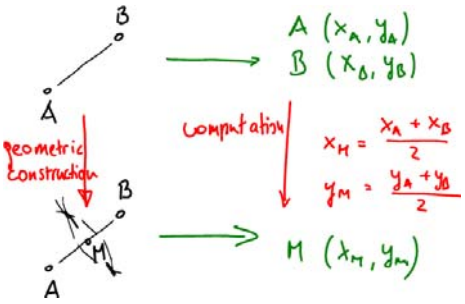


Figure 111: Homomorphism between construction and calculation

All the basic geometric constructions with ruler and compass have corresponding analytical operations. Therefore, all classical geometry can be redone with numbers, such that a homomorphism exists between the geometric construction and the analytical computation (Figure 111). The mapping is not an isomorphism, because the operations are not total: for some configuration, the computation fails because division by zero is not possible. To overcome this limitation is one of the goals of the following chapter 10xx.

2. Distance

Analytical geometry is based on coordinates, which describe points. Coordinates are distances for which a definition will be given here., Distance is the length of the shortest line between two points, it is finding the minimum. Distance is a function from two points to a positive real number, with three axioms (Figure 112):

(1) zero if the point is the same,
(2) symmetric: the distance is independent of order of the two points, and
(3) the triangular inequality.

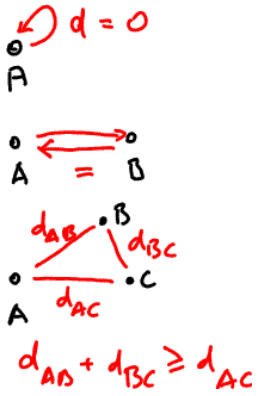


Figure 112: Distance relation

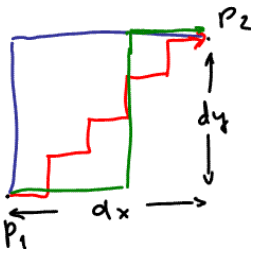


Figure 113: Manhattan or taxi-cab metric

These axioms do not uniquely define distance; many different formulae are possible; some examples are given, in the left column for the *2d* case and to the right, generalized for a space with *n*-dimensions. A circle is the geometric locus of all points with the same distance from a given point, different definitions of distance gives different "circles" (Figure 114).

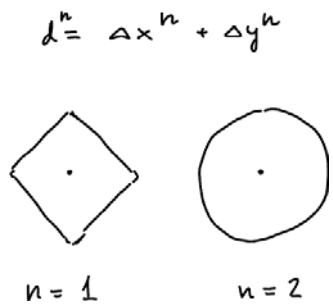


Figure 114: Circles for different distance definitions(Minkowski Norm n)

Euclidean Norm

$d = \sqrt{dx^2 + dy^2}$ $\quad d = \sqrt{\sum_v dv^2}$

Minkowsky Norm

$d = \sqrt[m]{dx^m + dy^m}$ $\quad d = \sqrt[m]{\sum_v dv^m}$

Cebysev Norm

$d = max\,(dx, dy)$ $\quad d = max\,(d_1, d_2, .. d_n)$

The Minkowski-Norm with *n=1* gives *d (dx, dy) = dx + dy* and is called Manhattan or taxi-cab metric (Figure 113). It gives the distance between any two points on a grid and is independent of the path:

Distance	
	dist (A, A) =0
symmetry	*dist (A, B) = dist (B, A)*
triangular inequality	*dist (A, C) <= dist (A, B) + dist (B, C)*

3. THE ALGEBRA OF VECTORS

Group (M, +, 0)
Operation: +, -
Rules: associative
(a+b)+c = a +(b+c)
Existence of identity
a+ 0 = 0 + a = a
Existence of inverse
(-x) + x = 0

Our experience with the manipulation of rigid bodies gives us some insight in the rules regulating operations with them: distance between points must be preserved; translations can be added, etc. From the formulation of such rules follow axioms for the operations with vectors. Vectors form an algebraic structure, called a *vector space*. We first give the algebraic structure in this section and then show in the following sections, how the axioms are justified by the geometric experience we have with rigid bodies.

*Ring (Q, +, *, 0)*
*A ring is a group with an additional operation, usually described as * which is distributive*
*a * (b + c) = a * b + a * c*
*(a + b) * c = a *c + b * c*

3.1 THE ALGEBRAIC STRUCTURE MODULE

A vector space is a *module* over a field which consists of two kinds of things: vectors, which are a commutative group (*M; +, 0*) and scalars, which form a ring with unit (*Q; +; *, 0, 1*). These

vectors and scalars are combined with an external operator scalar multiplication "·".

Note: The term "module" describes an algebraic structure and has no connection to the use of the same term in software engineering.

---

*Module <.> with group <M, +, >) and Ring with unit <Q, +, *, 0, 1>*
for all q, p,.. from Q and all a,b,... from M

$q \cdot (a + b) = q \cdot a + q \cdot b$

$(q + p) \cdot a = q \cdot a + p \cdot a$

$(q*p) \cdot a = q \cdot (p \cdot a)$

$1 \cdot a = a$

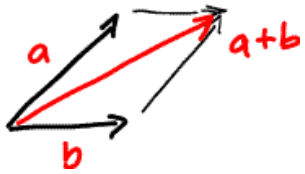
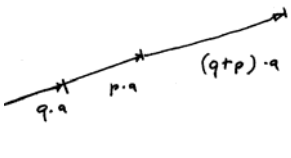


Figure 115: Addition of vectors

A vector space is a module where the scalars from a field (in practice: the real numbers)



$= q.a + p.a$

Geometric vectors form a vector space, i.e. a module over a field (usually the real numbers).

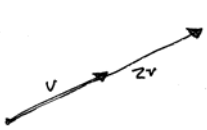


Figure 116: Geometric idea of vmult

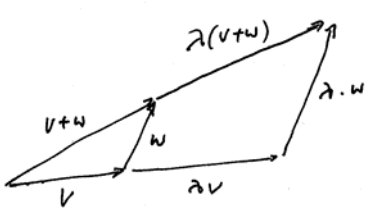


Figure 117: Multiplication is distributive over addition

3.2 *Left and right modules*

The scalar multiplication above was $q \cdot v$, which is a left module, because the multiplication with the scalar is from the left. A right module has the similar rules, but the scalar multiplication is $v \cdot q$. The right and left module are dual to each other; if the multiplication of scalars is commutative (which is the case for real numbers!) then $v.k = k.v$.

4. Geometric Interpretation of Vector Operations in *2* dimensions

Vectors are imagined as translation arrows in *n*-space. All vectors of the same length and direction result in the same translation, they are in one equivalence class. The zero vector has length *0*. Vectors are added by joining them geometrically (Figure 115); this construction is commutative ($a+b=b+a$) and the zero is a unit, that is, they form a group.

Multiplication of a vector with a scalar *s* extends the vector *s* times, keeping the direction (Figure 116). This multiplication is distributive over addition, etc. (Figure 117, **Error! Reference source not found.**, Figure 118).

5. Generalization: The Module of *n*-Tuples over R

The figures above were all for *2*-dimensional space, but the arguments are independent of dimension and valid for *n*-dimensional space. For *n*-tuples of scalars, we define a pointwise addition and a pointwise multiplication with a scalar:

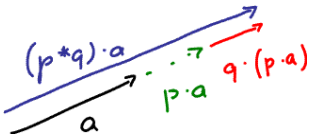


Figure 118: q . (p.a) = (q*p) . a

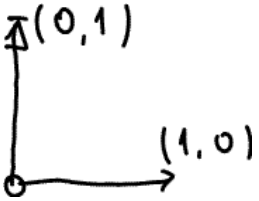


Figure 119: 2d base vectors

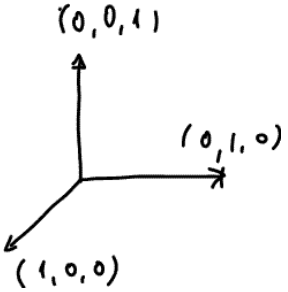


Figure 120: 3d base vectors

$(s + t)_i = (s_i + t_i)$
$(s * k)_i = (s_i * k)$.

Pointwise addition is commutative and has as a unit the *n*-tuple *(0,0,...0)*. Scalar multiplication is distributive over sum. The next section shows that these pointwise operations correspond to the geometric interpretation.

Note that this pointwise addition and multiplication with scalar for *n*-tuples is the same as the corresponding operations for polynomials. Polynomials form a module as well!

$p = p_1 * x_1 + p_2 * x_2 \ldots p_i * x_i = \Sigma\, p_i * x_i$
$p+q = \Sigma\, (p_i + q_i) * x_i$
$p * k = \Sigma\, (p_i * k) * x_i$.

6. SUBSPACES

Vector spaces show an important experience we have with space: they have subspaces and these are orderd by dimension. Two dimensional space is a subspace of 3 dimensional space. Subspaces are closed under the operations, a 2 dimensional translation remains always in the same 2 dimensional subspace (e.g. the surface of the table).

The dimension of a subspace is the minimal number of vectors which are necessary to span the space. The minimal set of vectors that span a subspace are linearly independent (as defined in the next subsection). Vector spaces with the same dimension are isomorphic, if we study one, we know them all!

7. POINTS IN SPACE: POSITION EXPRESSED AS COORDINATES

Coordinates are a mapping of points in *n*-dimensional space to *n*-tuples of scalars. The space is spanned by *n* base vectors (e_1, e_2,… e_n) and each *n*-tuple of coordinate values (This is often done in text books). from the field of reals corresponds to the point when multiplying the base vectors pointwise with the scalars in the n-tuple and adding them:

$v = (v_i)\ .\ (b_i)$ *where* $v = (v_1, v_2, \ldots v_n)$ *- the coordinate values*

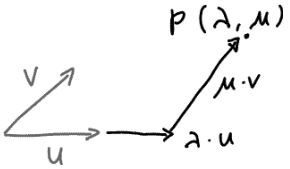


Figure 121: The mapping from (λ, μ) to a 2d point

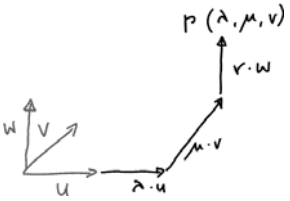


Figure 122: The mapping from (λ, μ, v) to a 3d point

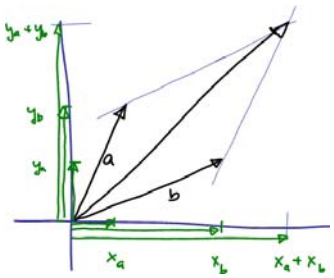


Figure 123: Addition is component-wise

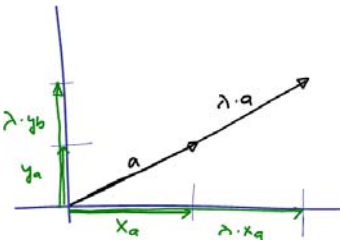


Figure 124: Multiplication is component-wise

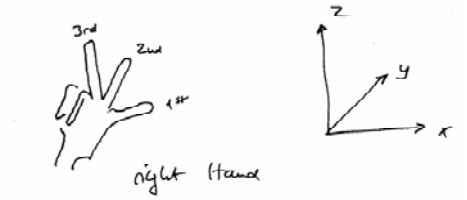


Figure 125: Right handed coordinate system

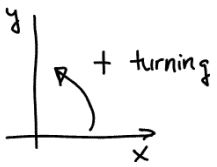


Figure 126: Positive turning

Positive turning direction: Mathematically defined (conventionally) as counterclockwise.

$e_i = (n_1, n_2 .. n_m) = (0, \ldots, 1, \ldots 0)$ *with* $n_i = 1$ *and all others 0.*

– the base vectors

$p = \Sigma\, vi * ei$ *- the vector with coordinates v*

The base vectors must be linearly independent. Linear independence means, there is no *n*-tuple of scalars s_i not all 0, such that sum $s_i * e_i = 0$; this is equivalent to say that the mapping from scalars s_i to vectors *v* have kernel 0. The *n*-tuples will be called coordinates of a vector; the mapping is isomorphic, and we identify an *n*-vector with the corresponding *n*-tuple (for a given basis) (Mac Lane and Birkhoff 1991, 195).

In general, the base vectors (unit vectors) need not be orthogonal and their length need not be the same. The orthogonal base for a vector space are the unit vectors (1,0), (0,1) or (1,0,0), (0,1,0), (0,0,1) in 2 respective 3-dimensions. The vector operations defined geometrically map to the corresponding operations on coordinates. Figure 123 shows how addition is done component-wise. Figure 124 shows that multiplication is equally component-wise.

8. Right Handed System of Vectors

A vector space is called right handed, if the vectors *x, y, z* in the order given are in a configuration like the first three fingers of the right hand. Mathematically, locking down the positive *z*-axis, turning the first coordinate axis towards the second axis in a positive direction, gives a right handed system.

Surveyors often use a left handed system, with north axis and east axis (north and easting as coordinates) and *z* (height) upwards. They measure the angles clockwise from North axis to east axis as positive:

9. Vector Is a Functor from Scalars to Points

The construction of vectors as tuples of scalars, typically real numbers, is a functor. It maps the scalars to vectors (tuples) for which the new operations obey the same axioms as for the scalars. This mapping is a (group) morphism, because it preserves the axioms. The unit of scalars maps to the unit of vectors and composition is the composition of the mapped values. Consider the special mapping, which maps every real *x* to the pair *(x,0)*. It is seen that this is a group isomorphism for plus and multiplication with a real maps to scalar multiplication.

$$(x,0) + (y,0) = (y,0) + (x,0)$$
$$= (x+y,0)$$
$$(x,0) + (0,0) = (x,0)$$
$$\vdots$$

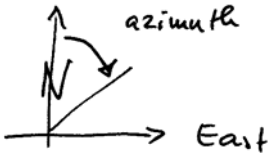


Figure 127: Geodesist use often left handed coordinate systems

Unitary operations, e.g., *+x,* are mapped to *+(x,0)*. The composition of an operation *+y* with another operation *+z* maps also and the identity operation is the mapping of *+0*:

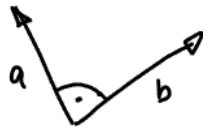
$$(x,0) + (y,0) + (z,0) = (x,0) + (y+z,0)$$
$$= (x+y+z,0)$$

10. VECTOR OPERATIONS

Our interpretation of vectors in space allows the expression of a number of geometric properties as operations on vectors. Three additional operations for vectors that have strong geometric properties are customary. They are useful to test for geometric properties and to give computational equivalent expression for geometric constructions(McCoy and Berger 1977, 433):

- inner (dot) product,
- cross product, and
- triple product.

This completes the program of analytical geometry: geometric properties are translated in algebraic properties and geometric operations are translated in algebraic operations. The defining axioms represent the geometric intuition and coordinates are not used for the definitions, but we show how the operations are translated to basic operations with coordinate values.



$a \cdot b = 0$

Figure 128: Orthogonal vectors

10.1 THE INNER (DOT, SCALAR) PRODUCT OF TWO VECTORS

The inner product of two vectors gives a scalar. Its definition is valid for all dimensions. For *2*- and *3*- dimensional space, it has interesting geometric properties. In geometry texts, the scalar product is written with a dot (but not the same dot as for composition of functions!), but sometimes it is written with brackets (<,>).

Inner (dot) product	*. :: vector -> vector -> scalar*
commutative	$a \cdot b = b \cdot a$
distributive	$a \cdot (b+c) = a \cdot b + a \cdot c$
a sort of associative law	$s\,(a \cdot b) = (s * a \cdot b)$
	$a \cdot 0 = 0 = b \cdot 0$

For tuples (a_1,.. a_n) the inner product is defined as pointwise multiplication

*(a_1, a_2, ... a_n) dot (b_1, b_2, ... b_n) = (a_1*b_1, a_2*b_2, ... a_n*b_n).*

This gives for example for the *2*-dimensional vectors previously introduced

```
(x1 y1) · (x2 y2) = (x1*x2, y1*y2).
```

Pointwise multiplication gives immediately the commutative and the distributive property from the corresponding properties of the multiplication in the ring of which the elements are formed from; associativity is also achieved.

$$s*(a \cdot b) = (s*a)\cdot b$$

$$LH \quad s*((a_1, a_2 \dots a_n)\cdot(b_1, b_2 \dots b_n)) =$$

$$s*(a_1*b_1, a_2*b_2, \dots a_n*b_n) =$$

$$s*a_1*b_1, s*a_2*b_2, \dots s*a_n*b_n)$$

$$RH \quad (s*a_1, s*a_2, \dots s*a_n)\cdot(b_1, b_2, \dots b_n) =$$

$$s*a_1*b_1, s*a_2*b_2, \dots s*a_n*b_n$$

10.1.1 Norm: the length of a vector

The *inner product* of a vector with itself is the square of its length and called norm |a|, which gives the ordinary Euclidean distance (Figure 129). It is easy to show that the norm satisfies the axioms for distances (a = 0, |a| = 0, |-a| = |a|, etc.)).

norm a = sqrt (a · a)

10.1.2 Angels between vectors

The inner product leads to the definition of angles between to vectors, which follows from the expression:

*a · b = |a| *|b| * cos (a,b)*

where (a,b) denotes the angle between the two vectors

*cos (a,b) = (a · b) / | a | * | b |*

Figure 129

One can interpret *a cos (a,b)* as the projection of the vector *a* onto the vector *b* (Figure 130), which links directly to the proof of the Cauchy-Schwarz-inequality (further reading on this important linkage between geometry and linear algebra see http://en.wikipedia.org/wiki/Inner_product_space).

10.1.3 Test for orthogonality

For two non-zero vectors, the inner product is zero if the two vectors are orthogonal (i.e., the angle between them is $\pi/2$), because then $v_1 = (x_1, y_1)$ and $v_2 = (x_2, y_2)$ where $x_2 = -y_1$ and $y_2 = x_1$. Orthogonality depends on the orthogonality of the base vectors.

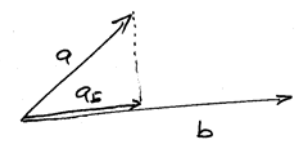


*Figure 130: a_b = |a| * cos (a,b)*

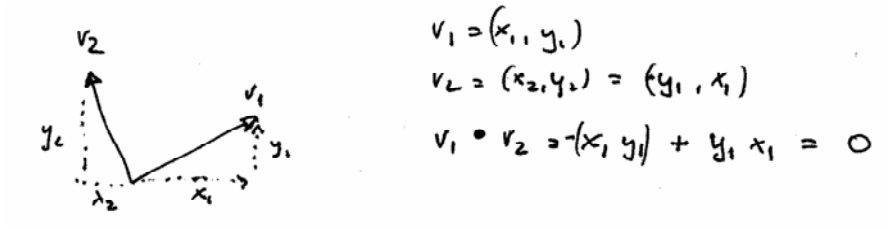


Figure 131

By definition, the 0 vector is orthogonal to every vector.

10.1.4 Unit vector in the direction of a given vector

It is convenient to compute a vector in the direction of a given vector but of known length (for example to compare the direction of two vectors).

```
unitVec :: m -> m -- a vector of unit length in direction of v
unitVec v = vmult (1/norm v) . v
```

10.1.5 Test for Parallel and antiparallel vectors

The dot product can be used to derive a condition for parallel and antiparallel (parallel but in opposite directions): the inner product divided by the product of the norms is for parallel vectors 1, for antiparallel it is -1 (Figure 133).

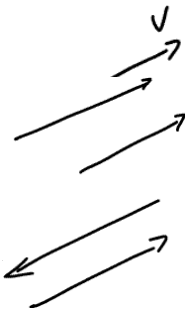
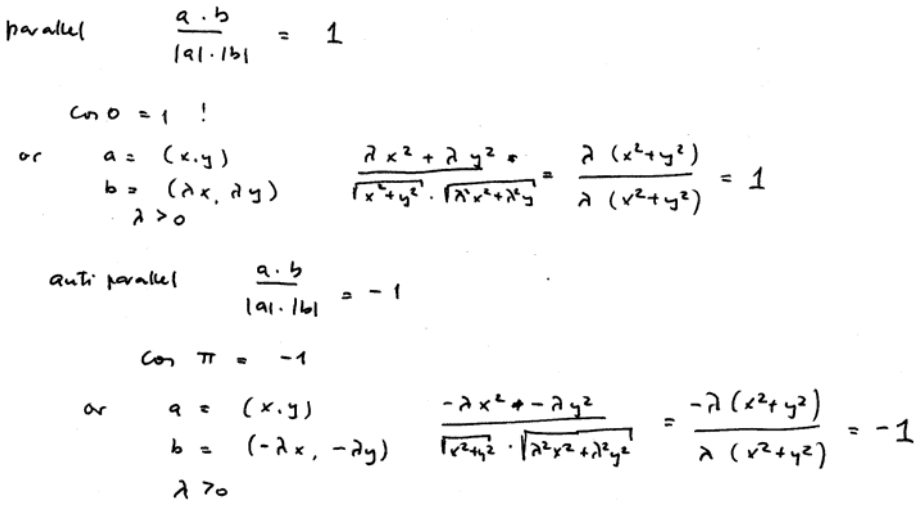


Figure 133: Parallel and anti-parallel vectors



10.2 VECTOR (CROSS, OUTER) PRODUCT FOR 3D SPACE

The operation cross product is defined only for spaces with *3*-dimensions. It takes two vectors and produces a vector. The vector product is a vector orthogonal on the two vectors and its length is the area of the parallelogram of the two vectors. The three vectors form — in a right handed vector space —a right-handed system.

A generalization for this operation available only in 3d space follows later to achieve dimension independent operations (chapter 20); the special case is useful for understanding the

construction in chapter 19 and connects with high-school mathematics.

Definition of Cross Product	*× :: vector -> vector -> vector*
	a× a = 0
anticommutative	*a × b = - b × a*
Distributive:	*a× (b + c) = a × b + a × c, (a+b) × c= (a × c) + (b × c)*
Sort of associative:	*s (a × b) = (s a) × b*

10.2.1 Definition *3d* vectors

For coordinates, the computation is

```
   (×) :: vec -> vec -> vec
(x1 y1 z1) × (x2 y2 z2) =
         (y1*z2 - z1*y2) (z1*x2 - x1*z2) (x1*y2 -
y1*x2).
```

This product derives from the regular multiplication of polynomials, if we assume the following equalities for products of base vectors:

$e_1 \times e_2 = e_3$, $e_2 \times e_3 = e_1$, $e_3 \times e_1 = e_2$ and $e_i \times e_i = 0$ for i= 1, 2 ...

10.2.2 Test for collinearity

a × *b* is zero, when *a* and *b* are collinear, in particular is *a* × *a* = *0*.

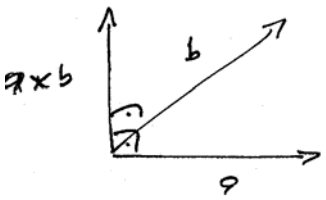


Figure 134

10.2.3 Area between two vectors

The area between two vectors in *3d* space is computed as (Figure 135)

area a b = norm (a × b) /2.

$$|a \times b|^2 = a^2 b^2 - (a \cdot b)^2$$
$$= a^2 b^2 - |a|^2 |b|^2 \cos^2 \varphi$$
$$= a^2 b^2 (1 - \cos^2 \varphi) = a^2 b^2 \sin^2 \varphi$$
$$\sin \varphi = \frac{a \times b}{|a| \cdot |b|}$$
$$a \times b = |a| \cdot |b| \cdot \sin \varphi \quad - \text{ area} \cdot 2 \; !$$

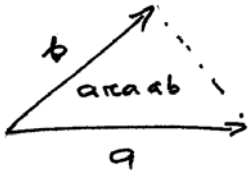


Figure 135: Area between two vectors

10.2.4 Collinear

Two vectors are collinear if the *vecProd* is 0; this does not depend on the orthogonality of the base vectors.

Collinear a b = varea a b == 0 -- or a× b == 0
Because sin α = 0

The 0 vector is collinear with every vector.

10.3 SCALAR TRIPLE PRODUCT (GERMAN: SPATPRODUKT)

This operation combines three vectors yielding a scalar. It is a combination of a cross product and an inner product. It is only

defined for vectors in *3d* space, but can be generalized for *n*-dimensional vectors (see chapter 20).

Triple Product	*Triple :: vec -> vec -> vec -> scalar.*
Cyclic permutations	*Triple a b c = triple c a b = triple b c a*

The scalar triple product is the same as the determinant of a 3 by 3 matrix (see chapter 10). It is signed. The triple product gives six times the volume of the parallelepiped of three vectors *a b c*. It is defined as:

$$\langle a, b, c\rangle = a \cdot (b \times c) = (a \times b) \cdot c$$

The triple product is zero if the three vectors are coplanar.

Coplanar a b c = triple a b c == 0

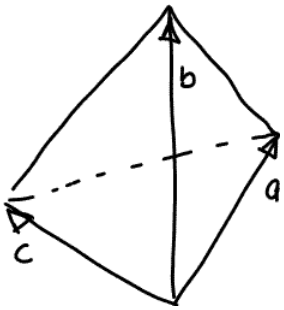


Figure 136: The volume of a pyramid

11. COORDINATE SYSTEMS

To establish a conventional (orthogonal) coordinate system requires an origin, a direction for the axis and a unit length for each dimension. the second axis is then orthogonal to the first and the unit vector has the same length (Figure 137). The values are of type length (meter in the SI system).

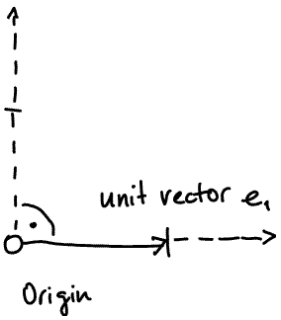


Figure 137: Definition of a coordinate system with origin, direction of axis and unit length.

The conventional geodetic system—WGS 84—takes the center of gravity of the earth as the origin, the direction of the rotational axis and the two orthogonal vectors are fixed such that one crosses the meridian of the old observatory in Greenwich (near London, UK). Most countries use local coordinate systems that are defined as projections from the earth surface to some convenient surface (cylinder or cones).

12. SUMMARY

The algebra for vectors is closed. The result of the operations is of the types other operations expect as inputs, which permits combinations of them in formulae of arbitrary complexity. This vector algebra for 2 and 3 dimensions generally used is the specialization of a vector a 2 or 3 dimensional subspace of a vector space. It is the result of constructing pairs or 3-tuples of real numbers with a functor.

The operations added to the standard operations of a vector space have useful geometric interpretation:

- Length of a vector is the norm (*sqrt (a · a)*),
- Area of between two vectors (*1/2 norm (a × b)*)
- Construction of a vector orthogonal to two given ones (a × b)
- Volume spanned by three vectors (1/6 triple (a,b,c))

- Collinearity and coplanarity (a x b = 0, triple (a,b,c) =0)

The operations are total (with the exception of computing the angle between two vectors).

Vectors form a vector space, which has a set of useful properties; geometric vectors provide a instructive example for a vector space, but the generalization is very fruitful and applicable in many situations which are not directly geometric. We will later use a vector space with functions (chapter 31) to describe movement of objects in time.

REVIEW QUESTIONS

- Demonstrate that pointwise multiplication of an *n*-tuple with a scalar *s* is distributive over addition.
- What is the difference between a right handed and a left handed system?
- Give definition of azimuth in geodesy?
- Explain dot and cross product. What are geometric interpretations, how is it computed?
- How do you determine if two vectors in a plane are orthogonal?
- How to compute the area of a triangle with vector operations?
- Why is the construction of vectors a functor? What needs to be demonstrated?
- What is meant by stating that surveyors use a left-handed coordinate system?
- What are the axioms for distance? Is the cost of a taxi ride a distance function? When is it? When not?
- Show that the cross product has the desired properties.
- Derive a formula for the computation of the area of a *2d* triangle given by the two vectors AB and AC.

Chapter 10 LINEAR TRANSFORMATIONS OF COORDINATE SPACE

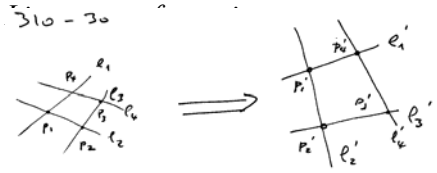


Figure 138: Incidence relations are preserved by linear transformations

Geometric transformation capture geometric properties(Blumenthal and Menger 1970); this chapter concentrates on *linear transformations* that transform straight lines into straight lines and preserve *collinearity* and incidence, that is, the intersection of two lines map to the intersection of the mapped lines. This is the geometry of projections. Linear transformations are part of linear algebra.

These transformations are expressed as vector operations, and do not access the coordinates directly, which demonstrate that the operations are independent of the details of the underlying coordinate systems. The discussion here is using examples from *2d* and *3d* space, but the result is independent of the dimension of the space; it applies to situations in a space-time continuum of 4 dimension or even higher dimensional spaces.

Matrices describe transformations

Matrices represent transformations of coordinate systems and a number of geometric problems can be expressed as transformations between different coordinate systems, including perspective projections. These are automorphism, they are morphism (mappings) from space to space. The introduction of matrix operations is motivated by spatial transformations (rotations). The purpose of the chapter is to describe the general linear transformation, such that transformation can be combined by multiplication.

Focus of chapter:
Automorphism of space.

The theory described here is applied when transforming or producing images in a GIS; for example the construction of the view of a landscape is using the projective transformation described at the end of the chapter. The inverse problem to construct a map given photographs is the domain of photogrammetry (Förstner and Wrobel Draft) and image processing; the material in this chapter is very similar to the foundations used today in modern treatment(Faugeras 1993; Hartley and Zisserman 2000).

1. LINEAR ALGEBRA—THE ALGEBRA OF LINEAR TRANSFORMATIONS

Linear algebra is among the best explored algebraic structures. Transformations that preserve collinearity between different coordinate systems are represented as linear transformation and can be represented as matrices.

Many geometric transformations are linear transformations. For example, stretching figures in one direction by a constant factor, reflection on a line through the origin, etc. are all linear transformations. They carry straight lines into straight lines, planes to planes, etc. or more generally, geodesics transform to geodesic, such that incidence is preserved: the transformed intersection point is the intersection point of the transformed lines.

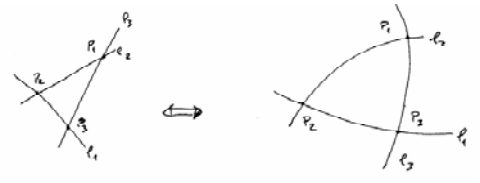


Figure 139: General linear transformations preserve collinearity

The composition of two linear transformations is again a linear transformation. This chapter shows, how composition of linear transformations can be computed as multiplication of the single transformations (Figure 140). This chapter shows the different transformation matrices that correspond to the typical linear transformations like translation, rotation, perspective projection, etc..

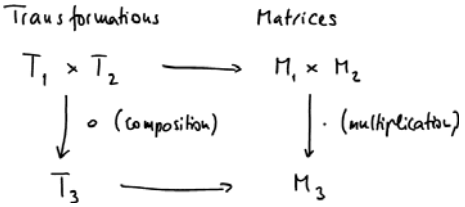


Figure 140: Composition of Linear Transformation maps to multiplication of matrices

Multiplicative transformations must leave invariant the origin of the coordinate system; it is not possible to combine translations—which move the origin of the coordinate system—with rotations and other transformations directly. We extend the representation and go from *n*-dimension to *n+1*-dimensions. This chapter will introduce these so-called homogenous coordinates and we will see that the transformation from ordinary to homogenous coordinates is a functor.

2. LINEAR TRANSFORMATIONS

Linear independence means that there are no scalars a,b,c not equal zero for which

$$a \cdot u + b \cdot v + c \cdot w .... = 0$$

A linear transformation is any transformation *t: R -> R'* which is an automorphism between *R* and *R'*, is additive and homogenous (*k* is a scalar, *t* is a linear transformation)

$t(a + b) = t(a) + t(b)$ *additive*
$t(k * a) = k * (t(a))$ *homogenous*

The treatment here uses the right multiplication for the scalar (see chapter 9) alternatively a left multiplication with the same rules is possible and defines the dual algebra, which will be used later in chapter 19. For vector spaces over commutative rings, and the real numbers form a commutative ring, the left and the

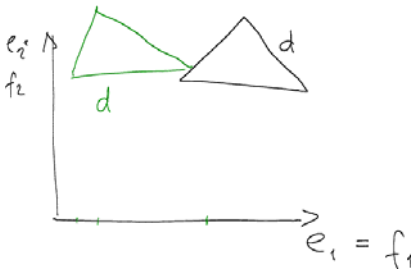


Figure 141: The image of the same figure before and after transformation

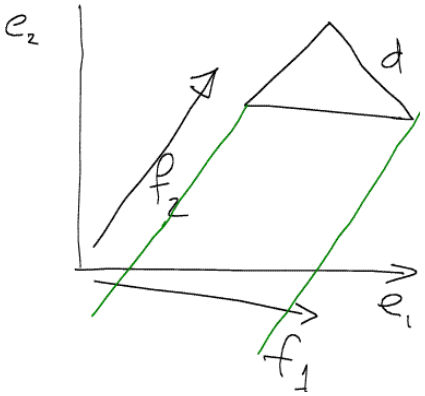


Figure 142: The same figure in two coordinate systems

Think of a matrix as a transformation!

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1m} \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2m} \\ \vdots & & & & \\ a_{n1} & a_{n2} & a_{n3} & \cdots & a_{nm} \end{bmatrix}$$

Figure 143: A matrix

[3.2] ≠ 3.2

right module is isomorphic. We nevertheless stress this duality which maps directly to the duality between points and lines used in the next chapter.

3. TRANSFORMATIONS OF VECTOR SPACES

Linear transformations are usually seen as a transformation of a figure (Figure 141) but an alternative view of the transformation as a transformation of space is possible: A vector describes a point as a list of scalars to multiply a list of base vectors with; selecting different base vectors a different vector results for the *same* point. This is a transformation of space to itself: to every point belongs a set of coordinates with respect to the first and to the second set of base vectors. This is difficult to visualize, because the point remains the same (Figure 142), but we can imagine that the base vectors remain fixed and then see where points are mapped (Figure 141).

4. DEFINITION OF MATRIX

Linear transformations are important enough to warrant an algebra, the algebra of matrices. A matrix represents a transformation between two vector spaces, each with a base. The columns of the matrix of a transformation are the transforms of the unit vector from the base (see Figure 151).

A matrix can be seen as a function from indices to a scalar value (Figure 143):

m :: Int -> Int -> Scalar.

It is written as $A = [a_{ij}]$.

Matrices form a vector space, i.e. a special module (see chapter 9xx).

4.1 DIMENSION

The dimension of a matrix is the number of rows and columns. Mind, that a matrix with dimension 1 by 1 is not the same as a scalar value. One is a matrix with one element, the other is a scalar!.

4.2 POINTWISE DEFINITION OF ADDITION AND SCALAR MULTIPLICATION

Addition of two matrices of same dimension is pointwise sum (like for vectors) and the multiplication with a scalar is the multiplication of each element by the scalar.

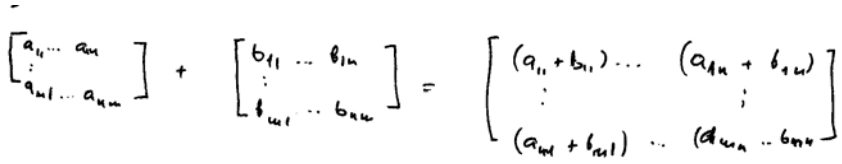


Figure 144: Σ aij + bij

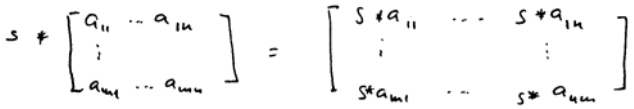


Figure 145: Multiplication of matrix with scala: $(s * a)_{ij} = s * a_{ij}$

4.3 UNIT MATRICES FOR ADDITION

The zero matrices (the units for addition) are the matrices with all elements equal zero; they are written as 0 but there exist a different zero matrix for every dimension.

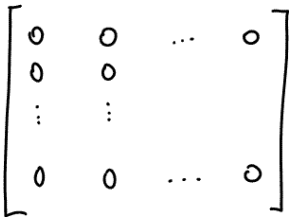


Figure 146: The zero matrix

4.4 MATRIX MULTIPLICATION

The multiplication of two matrices producing a matrix is a new operation; written as "*", but often without a symbol, *AB* meaning the multiplication of *A* with *B*.

```
matMult :: mat -> mat -> mat
```

The multiplication of two matrices is defined as the inner product of the column and row vectors in all combinations; it is only defined if the number of rows of the first matrix is the same as the number of columns of the second one. This definition assures that the composition of linear transformation is multiplication of the corresponding matrices (Mac Lane and Birkhoff 1991, p. 225).

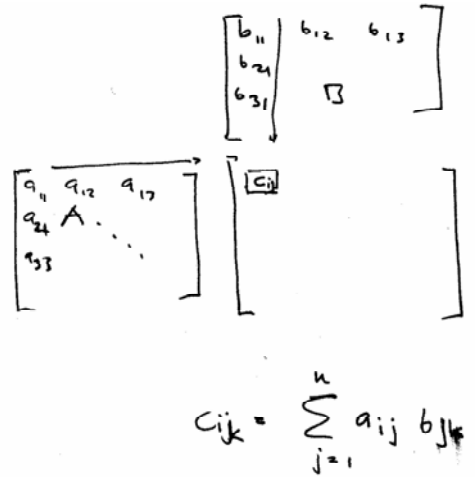
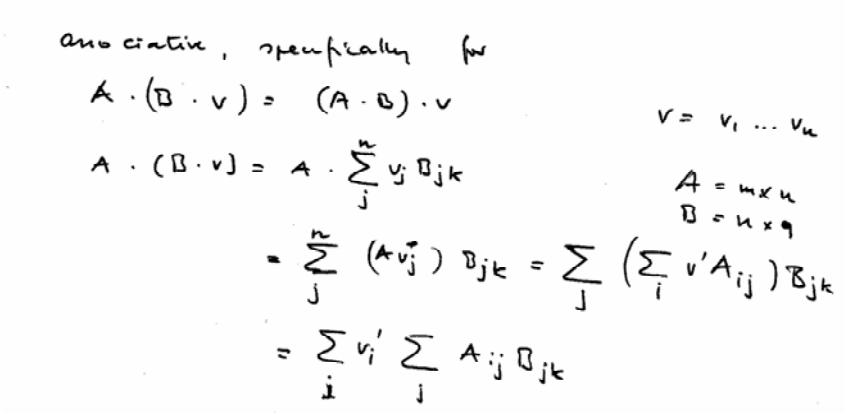


Figure 147: Multiplication of two matrices

This multiplication is associative, but not commutative $A\,B \neq B\,A$.



4.5 UNIT FOR MULTIPLICATION

The unit matrices for the multiplication are the a square matrices I with all ones in the diagonal and the non-diagonal elements equal to zero. Then $A * I = A$ for any A (with the right dimension for I). Unit matrices can be constructed for any dimension. One can think of the I matrix as a collection of the base vectors for this dimension. The unit matrix I can be defined with the Kronecker δ:

$i_{ij} = \delta_{ij}$

where $\delta_{ik} = 1$ for $i=k$ and 0 for $i \neq k$.

$$\begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & & \\ \vdots & \vdots & & \\ 0 & 0 & \cdots & 1 \end{bmatrix}$$

Figure 148: One matrix

4.6 TRANSPOSE

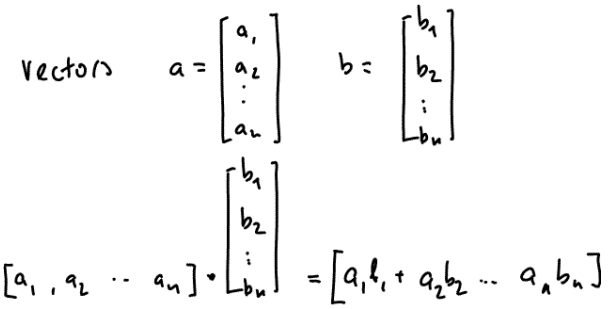
The transpose of a matrix is the matrix with rows and columns exchanged. The transposed matrix is the mirror image around the diagonal.

$A^T{}_{ij} = A_{ji}$

For transposed matrices $AB = B^T A^T$ applies

$A * v = t \rightarrow v^T * A^T = t^T$.

From these definition follows, that we can use a given a matrix multiplication to multiply vectors: If we think of vectors as a matrix of a single column, then we have to transpose the first matrix before the multiplication: $a \cdot b = A^T B$.



4.7 RANK

The rank of the matrix corresponds to the dimension of the vector space that is spanned by the matrix, taken the columns as

base vectors. It counts how many linearly independent vectors it consists of. The rank of a matrix and the rank of its transposed is the same; the rows can be considered to check rank as easily as the columns. The rank of a matrix is the same as the dimension of the vector space it spans.

rank (A) = rank (A^T)

4.8 DETERMINANT

The determinant is a *multilinear*, alternating form. It is zero if any two rows or columns are linearly dependent on each other. The determinant of a 2 by 2 or 3 by 3 matrix is computed as the sum of the products along the main diagonals minus the sum of the product along the minor diagonal.

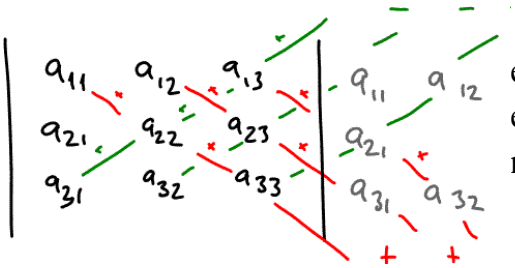


Figure 149Determinant as sum of products of main diagonal minus product of minor diagonal

The determinant of larger matrices is by recursively expanding it to a column (row) to the alternating sum of the elements of this column times the smaller determinant of the matrix with this column and row crossed out (Figure 150).

$$\det \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} =$$

$$a_{11}\, a_{22}\, a_{33} + a_{12}\, a_{23}\, a_{31} + a_{13}\, a_{21}\, a_{32} - a_{11}\, a_{32}\, a_{23} - a_{21}\, a_{12}\, a_{33} - a_{31}\, a_{22}\, a_{13}$$

$$= a_{11} \cdot \det \begin{bmatrix} a_{22} & a_{23} \\ a_{32} & a_{33} \end{bmatrix} - a_{21} \cdot \det \begin{bmatrix} a_{12} & a_{13} \\ a_{32} & a_{33} \end{bmatrix} + a_{31} \cdot \det \begin{bmatrix} a_{12} & a_{13} \\ a_{22} & a_{23} \end{bmatrix}$$

Figure 150: Determinant of 3 by 3 matrix

The scalar triple product is the determinant of a 3 by 3 matrix constructed from the joining of the three vectors.

tripleProd a b c = (a . (b x c)) = det [a,b,c]

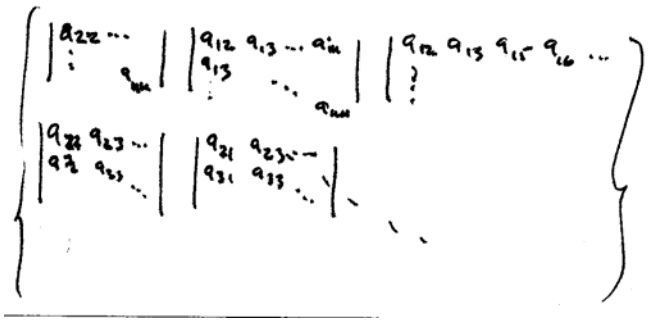
Square matrices

det A = det (transp A)
*det (A * B) = det A * det B*
*det k * A = k^n * det A (where n is the dimension of A)*

4.9 COFACTOR AND ADJOINT MATRIX

In general, a matrix such that each entry is the value of the determinant of the original matrix with the row and column of the element crossed out, is called the cofactor matrix.

```
Cof :: matrix -> matrix.
```



The transposed of the cofactor matrix is called the adjoint.

$a \cdot cof(A)^T = det\ A \cdot I$

$adj\ A = transp\ (cof\ A)^T.$

4.10 Inverses Matrix

Square, non-singular, matrices have inverses, such that:

$A\ A^{-1} = I.$

A square matrix is singular if its rank is less than its dimension. The determinant of a singular matrix is zero. The inverse can be computed as the adjoint matrix multiplied with the inverse of the determinant of the matrix, which is a scalar (never 0 for a non-singular matrix!)

$a^{-1} = (cof\ a)^T * (1/det\ a)$

4.11 Orthogonal and orthnormal matrices

Matrices where all the row vectors are orthogonal (i.e., the pairwise inner product equal zero) are called orthogonal. If all the vectors have length 1, then the matrix is normal. For orthonormal matrices, matrices with orthogonal and normal vectors, the determinant is either 1 or –1. The inverse of an orthonormal matrix is the transposed. The product of orthogonal matrices is again orthogonal.

4.12 Elementary Operations and Equivalence of matrices

Two *m x n* matrices A and B are equivalent if there is a sequence of elementary operations on rows and columns carrying A to B (Mac Lane and Birkhoff 1991, 225-229). Elementary operations are:

- Exchanging two rows (or column)
- Multiplying a row (or column) by a scalar
- Adding a multiple of one column (or row) to another column (or row)

Matrix	
Not commutative	$A * B \neq B * A$

*(A * B) T = BT * AT*

The effects of elementary operations on the determinant are:

- Exchanging two rows (or columns) multiplies the determinant by –1 (alternating form)
- Multiplying a row (or column) by a scalar multiplies the determinant by the same scalar (multilinearity)
- Adding a multiple of one column to another column (or row) leaves the determinant unchanged.

5. TRANSFORMATIONS BETWEEN VECTOR BASES

The problem of transforming coordinates (p_u, p_v) expressed as factors to a list of base vectors u, v… into the coordinates (p_x, p_y) as factors to a list of base vectors x, y .. , requires that we have the coordinates for the vectors u, v in the system given by x, y. These are $u = (x_u, y_u)$ and $v = (x_v, y_v)$. The transformation between coordinates expressed in different base vectors, but with the same origin, is for the *2d* case:

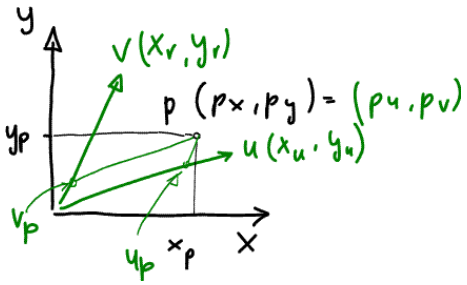


Figure 151: Transformation of a vector from x-y to u-v coordinate system

$$p_x = p_u \cdot x_u + p_v \cdot x_v$$

$$p_y = p_u \cdot y_u + p_v \cdot y_v$$

$$\begin{bmatrix} p_x \\ p_y \end{bmatrix} = \begin{bmatrix} x_u & x_v \\ y_u & y_v \end{bmatrix} \cdot \begin{bmatrix} p_u \\ p_v \end{bmatrix}$$

Given p in system uv

Find p' in system xy

1. find u', v' in system xy: (x_u, y_u), (x_v, y_v)
2. write them as columns $T = [u', v']$ *gives* $p = T * p'$
3. invert matrix; this is the transformation matrix: $p' = T^{-1} p$

Proof: $T\,[u,v] = [u', v']^{-1}\,[u,v] = I$

This is a linear transformation where the base vectors expressed as coordinates of the new base give the inverse transformation matrix. The multiplication of the matrix was defined as the multiplication of a sequence of vectors with the point vector to be transformed (*dot product*). The vectors in the matrix are the coordinates of the old base vectors in the new base. This justifies the definition for matrix multiplication introduced before.

Note:
The transformation of a vector *v* by a matrix *M* is written as *M v*, similar to the transformation of a value by a function (*f x*). The vector *v* is a column vector. Some texts use the alternative notation of $v^T M^T$, multiplying the row vector by a matrix from the right. In this case, the matrix is the transposed matrix to our notation.

6. LINEAR TRANSFORMATIONS FORM A VECTOR SPACE

Translations, rotation, perspective projections, etc. are linear transformations. We have seen before that these transformations form groups, but they also form a vector space!

6.1 TRANSLATIONS

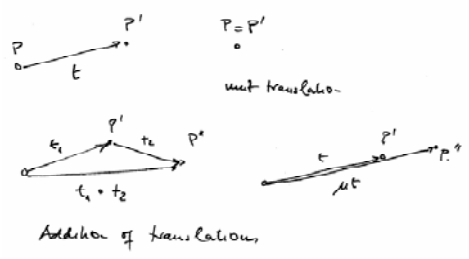


Figure 152: Addition of translation

two different interpretations for a vector:
as a point
as a transformation (translation).

Translation of a vector by a translation vector is vector addition (pointwise addition). The operations for translations are the same operations than for vectors and we can identify the translations and the corresponding translation vectors. Translations form a vector space. This can be 'unified' to an understanding of 'Each vector represents the point, to which the origin is translated with this vector". The operations are the same for both interpretations.

A translation cannot be expressed as a matrix multiplication, because a matrix multiplication is a group isomorphism. The translation is a bijective mapping (see chapter 5) but not an isomorphism of vector space. Observe that the *0* vector is mapped by a translation *F (t)* to the vector *t*. This violates the condition for an isomorphism, where the *0* must be mapped to the *0* (the kernel of *F (t)*) must be the unit.

f (a + 0) = f (a) = a + t
f a + f 0 = a + t + t

Universal mapping property for an isomorphism:
Kernel f = {units}

6.2 ROTATIONS

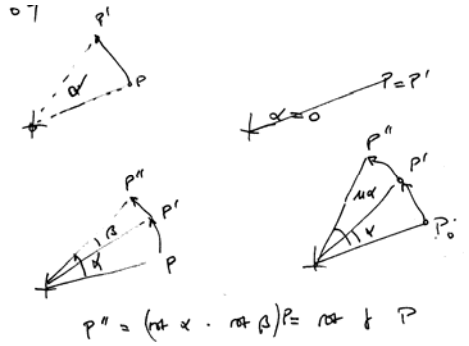


Figure 153: Rotation

Kernel R = {0}

Rotations are a group and they form, a vector space. The rotation of a vector by an angle alpha results in a vector:

$$x' = R \cdot x = \begin{bmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

The composition of rotations is just matrix multiplication: *R1 (R2 v) = (R1 * R2) v*. Rotation is a bijection, which is an isomorphism; it maps the *0* to the *0*. We see here that in a vector space, the origin (the unit) plays a special role (Mac Lane and Birkhoff 1991).

6.3 SIMILARITY AND AFFINE TRANSFORMATIONS

The general similarity transformation has 4 degrees of freedom, scale, rotation angle and two translation values. It can be written as a matrix followed by a translation with a vector $t= (t_x, t_y)$.

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} s \cdot \cos\alpha & -s \cdot \sin\alpha \\ s \cdot \sin\alpha & s \cdot \cos\alpha \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

The affine transformation is composed of a non-uniform scaling by a non-singular 2 by 2 matrix followed by a translation and rotation. It cannot be expressed as a single 2 by 2 matrix. The similarity transformation has 6 degrees of freedom:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

7. GENERAL LINEAR TRANSFORMATIONS

Can we unify all the transformations in a single framework? All transformations preserving collinearity should compose to form other transformations which again preserve collinearity. The general linear transformations form the 'General Linear Group' *GL(n,F)*, where *n* is the rank and *F* the field over which the transformations are constructed; the field is in this case usually the field of real numbers. They are the invertible, non-singular matrices of size $n \times n$. It is isomorphic to the group of automorphism of any *n*-dimensional vector space *V* over *F* (Mac Lane and Birkhoff 1991, 247).

The general linear group for *n* does not include all the transformations in *n*-dimensional space preserving incidence we are interested in. A similarity transformation preserving incidence for a space of dimension *v* could be written as a rotation and a translation: $x' = A \cdot x + b$, where *A* is a matrix of dimension $v * v$, *b* a vector of dimension *v*. This transformation, expressed in the form of translation and rotation, cannot be composed through matrix multiplication, because they do not leave the origin at the same place. Composition of transformation by matrix multiplication means that a transformation composed of first t_1 and then t_2 is the single transformation $t_{12}= t_1 t_2$, which can be applied to all points *p*. This is only possible, if $t (0) = 0$.

In order to bring translations and rotations—and some other transformations—in a single system, we have to add a dimension (from *2d* to *3d*, from *3d* to *4d*) . We keep the origin of this higher

dimensional space fixed, but move the point to which the origin of the space of interest maps to. For these homogenous vectors, the transformations map the origin to the origin; this means going to the projective space. We map *n* vectors to the corresponding homogenous *n+1* vectors with a functor.

The *2d* plane of interest is embedded into a *3d* space as shown in Figure 155. A line through the origin and all its points are the equivalent representations of a point in the *2d* plane (so called homogeneity). This is a interpretation of the projective plane, which we will introduce later (chapter 19). Adding one dimension *w* is sufficient to achieve the purpose of composition of linear transformations by matrix multiplication:

N (M a) = (N matmult M) a.

Consider the plane of the *w* axis and point *p* (Figure 156). The translation of *p* by *t* becomes a rotation followed by a change of scale—and scale transformation can be ignored, because p_1 is (homogenous) equivalent to *p'*.

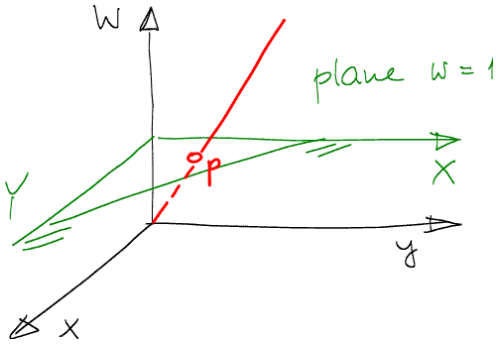


Figure 155: The point p and all points equivalent in homogenous space

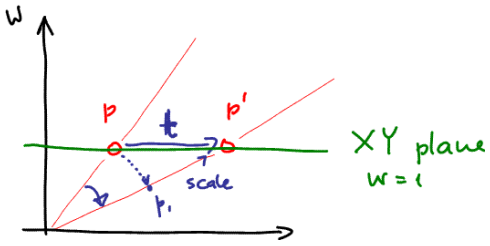


Figure 156: Translation becomes a rotation and a scale change

Homogeneity:
The algebraic entity a is called homogenous if a and λa, with λ≠0 represent the same geometric entity (Förstner and Wrobel Draft)

7.1 HOMOGENOUS COORDINATE SYSTEM

Homogenous coordinates were invented by Maxwell (1831-1879) to have a well-behaved algebra for geometric objects, points, lines, areas, and transformations. Note, that *2d* vectors do not behave nicely—remember that cross product is not defined, but *3d*, *4d* (so-called quaternions, often used in geodesy for *3d* space and time) and *8d* vectors allow definitions for a multiplication with (some of) the regular properties, in *3d* space this is the cross product.

Homogenous coordinates were used in computer graphics (Newman and Sproull 1981; Foyley and van Dam 1982) because they avoid divisions, which were with the hardware of the 1970s and 1980s much more time consuming than additions and multiplications. All divisions in a computation are collected in the scale factor that is applied only at the very end. This performance consideration is not important today, but the same property makes homogenous coordinates helpful to construct total functions: they avoid divisions, and divisions are an important place where functions become non-total! The purpose is to write equations for total functions in homogenous coordinates—that is, using the projective plane—where ordinary formulae would yield non-total functions.

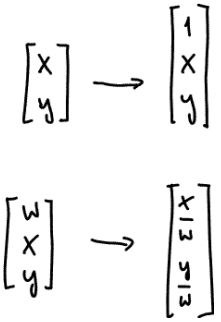
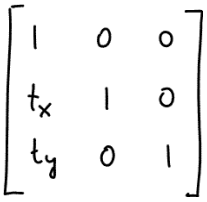
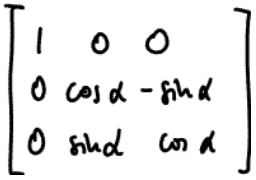


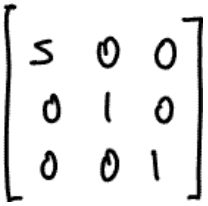
Figure 157: Transformation from 2d to homogenous and homogenous back to 2d coordinates.



Translation



Rotation



Scaling

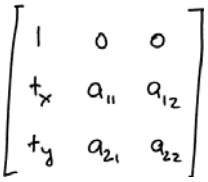


Figure 158: Affine transformation

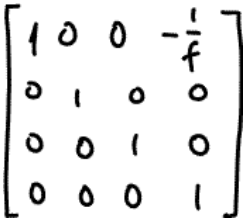


Figure 160: Perspective Transformation

7.2 Mapping from regular 2D to homogenous 3D coordinates

The transformation of regular *2d* coordinates to homogenous coordinates is by adding the homogenous coordinate with $w = 1$. The transformation from homogenous to Euclidean *2d* is by dividing the x and y values by the (scale factor) w.

Many texts add the 'homogenous coordinate' (the 1) at the end of the vector. To prepare for a dimension independent formalization, we have the first element in the vector represent the homogenous value. This mapping is a functor.

7.3 Transformations

The transformations we have seen before can all be expressed as matrices in homogenous coordinates. Complex transformations, like similarity and affine are composed by multiplication. But now we can also include scale changes and translations and even perspective projection can be expressed (as a mapping from 3-dimensional space to a plane in 3-dimensional space): Assume that the optical plane of the lens is in the x_1, x_2 plane(Stolfi 1991p. 74; Förstner and Wrobel Draft):

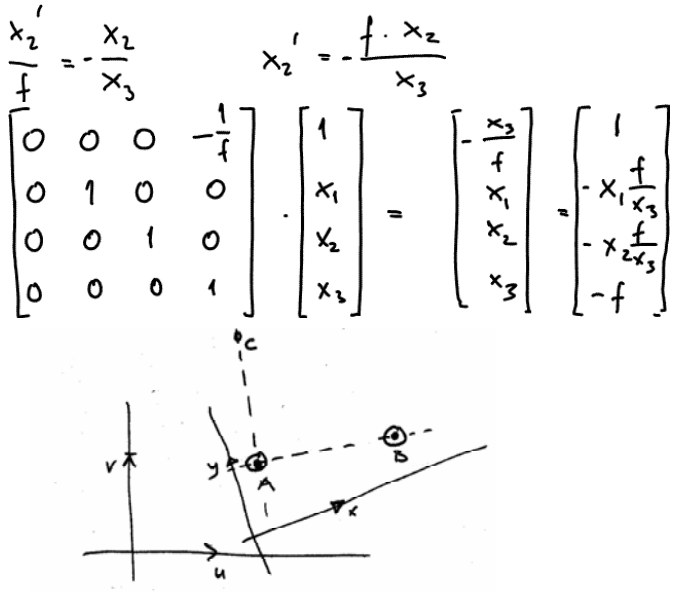


Figure 159:Transformation of points in 2d

8. Special Case: Similarity Transformations in *2D*

The determination of a transformation of three points given in two coordinate systems (a,b,c and a', b', c') is

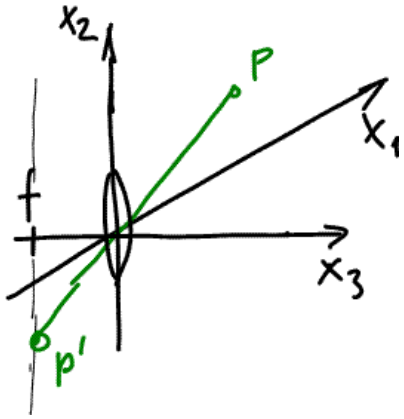


Figure 161: A lens with focal length f and a point with its image

$$a' = T\,a$$
$$b' = T\,b$$
$$c' = T\,c$$

$$[a', b', c'] = T\,[a, b, c]$$

$$[a', b', c']\,[a, b, c]^{-1} = T \cdot I = T$$

A similarity transformation preserving angles is determined by only 2 points. The transformation matrix has the form:

$$\begin{bmatrix} s & 0 & 0 \\ t_x & \cos\alpha & -\sin\alpha \\ t_y & \sin\alpha & \cos\alpha \end{bmatrix}$$

With a parameter for rotation, two for translation, and one for scale. The general approach in section 5xx does not work, because we have only 2 points. We can either add constraints to the system of equations or construct a third point C such that C – A – B is a right angle (Figure 159) and compute the coordinates of C in both systems. Then we have 3 points and can use the general formula.

9. SUMMARY

In this chapter the unification of different transformations were achieved using homogenous coordinates that are a representation of projective space. Adding one dimension to the vector space it was possible to achieve a simple, unified framework. Transformations form a category, where composition is defined. This is again the construction of a functor to expand a representation when it is insufficient to represent all cases.

The chapter also showed how to overcome the limitations that some operations of vector algebra are restricted to *3*-dimensional spaces. The transformation formulae are using only matrix operations, which are valid for square matrices of any dimension.

REVIEW QUESTIONS

- Why are homogenous coordinates necessary? Give transformations between homogenous and orthogonal coordinates in both directions.
- Explain the formulae for transformations (translations, rotation, perspective transformation) using homogenous coordinates.

- Why are homogenous coordinates allowing us to combine all different transformations into a single general linear transformation?
- What is a general linear transformation?
- Demonstrate that translation and rotation leave distances invariant.
- Show that the transformation from ordinary 2 vectors to homogenous 3 vectors is a functor.

PART FOUR FUNCTORS TRANSFORM LOCAL OPERATION TO SPATIAL AND TEMPORAL DATA

Observations produce measurement of different types which are combined in functions to produce values of interest (see chapter 6). Soil type, exposure, annual rainfall and similar locally observed values are combined, for example, in a formula to compute the agricultural value of land or the potential for soil erosion. These formulae express relationships between values valid at a single point in space and time.

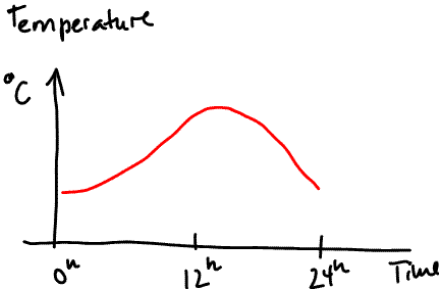


Figure 162: Temperature in function of time

In this part in chapter 11 and 12, we show first how such formulae can be applied in a principled way to time series (Figure 162) and to spatial layers (Figure 163) of point data values. We use here the methods to represent points in space and time given in the previous part and treat the observed values at these points, which we call "point data". Time series of observed values can be combined to show how a computed value changes with time. A formula to compute the values for a point can be applied to a layer of similar measurements and produce a map showing how the value changes in space.

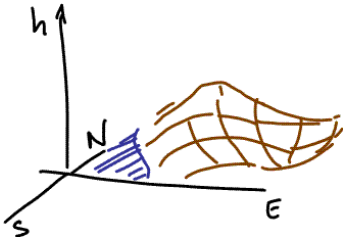


Figure 163: The surface of the earth as a function of position

Functors are the mechanism to expand local functions to apply to layers of spatial data and time series (chapter 6.4xx). A functor is a morphism, which preserves composition and identity. It is an often used method to construct new algebraic systems from given ones. The functors introduced here expand the domain of application of functions from local application to values observed in space or time, or even space-time.

Map Algebra is a part of GIS theory that has survived for more than 20 years without much change(Tomlin 1983); it will be formalized and generalized in this part, but not altered in a substantial way. At the core of map algebra is a functor *layer*, but Map Algebra includes more operations than just the local operations produced by this functor. Tomlin identified

- *local operations* (chapter 12) characterize a location,
- *focal operations* characterize a location within its neighborhood (chapter 13).
- *zonal operations* characterize a location within the area of similar properties, its zone; they have a different structure and link towards the identification of objects in space or events in time (chapter 14).

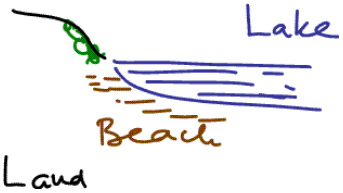


Figure 164: Beach is the zone where the lake influences the land

Focal operations are an example of *convolution*. Convolutions are mathematically well defined and extensively used in image processing(Horn 1986). They are not just useful for image processing of remote sensing data, but are a method to analyze geographical situations stressing the concept of "neighborhood": the properties of the areas immediately around a location influence this location. For example, a lake influences the land in its vicinity—and this influence is important enough that we have a special word for it, namely "beach" (Figure 164. A local operation is not sufficient to find beach areas, it is necessary to have an operation that considers the neighborhood: a beach is where water and land meet. Tomlin called this *Focal Operations* and included in this class all operations which are considering values in the immediate neighborhood.

In this part, the application of operations to time series and to spatial map layers is unified in the same conceptual framework. The treatment of time series is presented first (chapter 11), because the graphical presentation is simpler and the following generalization to *2*- and *3-dimensional* space and the combination to spatio-temporal data are straightforward.

Chapter 11 FLUENTS: VALUES CHANGING IN TIME

Values that change in time, for example the outside temperature, are common examples to demonstrate the treatment of data that represents observations varying with time. If we observe inside and outside temperatures, we can compute for any point in time the difference between them. The values changing in time can be seen as functions from time to a value, in this special case a function from time to a temperature value. Operations, e.g., difference, can be applied to such functions and return a function 'difference between inside and outside temperature'.

Functors apply to operations with values and produce functions which take time series of values as inputs and produce time series. With functors simple operations on values become polymorphic and apply equally to time series and map layers (see next chapter).

Τὰ πάντα ῥεῖ.
All is flow
Heraclites

1. CHANGING VALUES IN TIME

Life is change; everything in the world is in flux. In this chapter we concentrate on point observations at the same location repeated in time, but change is also affecting the properties of objects, which will be discussed later. Observations that return different values varying with time are common and can be contrasted with values assumed to be constant. McCarthy and Hayes called properties that change *fluent(McCarthy and Hayes 1969)*.

Fluent = value that changes with time

The values shown in Table 2 give the observed temperature at a fixed location inside and outside of a building during a day.

Time	*Inside temperature °C*	*Outside temperature °C*
7:00	18	5
8:00	21	7
9:00	21	8
10:00	20	10
…	…	…

Table 2: Temperature readings inside and outside of a building

Values describing properties of objects change in time, some rapidly, some very slowly. Only very few natural constants, e.g., the Boltzman constant, do not change. Some changes are so slow compared to the focus that they are not relevant (Figure 165) and

First Law of Time:
Everything changes, but some things change slower (and are constant relative to the faster changing ones).

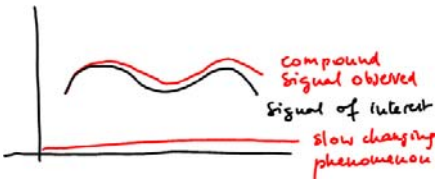


Figure 165: Slow changing, quasi constant phenomenon

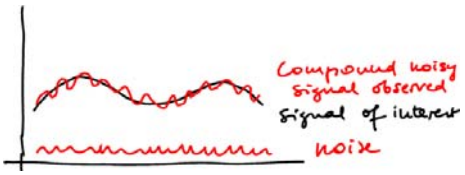


Figure 166: Signal and noise

Second Law of Time:
Some changes are so fast that they appear as noise compared to slower changing things.

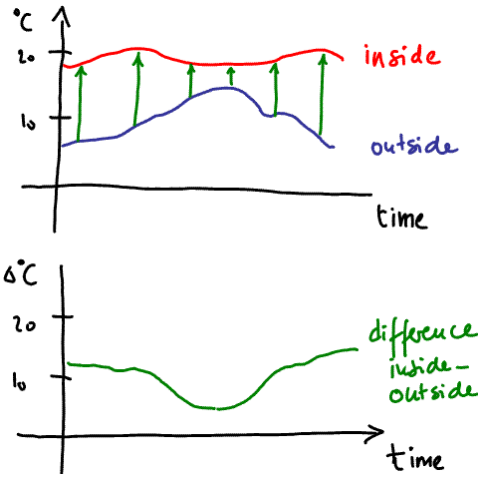


Figure 167: a) Outside and inside temperature, b) difference

others are changing so rapidly that they appear as noise compared to the signal of interest.

The treatment of values that change is difficult in first order languages: values in a formula are constants and the result of a computation is a (constant) value. To gain a handle on temporal data, we must use a second order language (see chapter 4.5), where variables can be functions. This approach is powerful and leads to a formalized treatment. The alternative is *situation calculus(Lifschitz 1990)*, which is based on first order logic but still requires second order, extra-logical operations(Reiter in preparation).

2. Synchronous Operations on Fluents

Assume we measure the temperature inside and outside, then we may also compute the difference between inside and outside for any point in time (Figure 167). This difference exists for any point in time: $d(t) = i(t) - o(t)$.

We call such computation '*synchronous'* because the values corresponding to the same time instant are combined in a computation. The computation $d = i - o$ is *inside a snapshot*; the computation is not dependent on the time. Every operation inside a single snapshots can be extended to a calculation on the corresponding number of changing values. The functor *fluent* is a mapping from values of type *Float* to functions resulting in a value of type *Float* with the signature *t -> Float*.

3. Fluents Are Functions

A fluent is a function from time to a value. Operations for fluents are defined as the synchronous application of the operation for each time point. For example, the difference between the two functions inside and outside temperature is a function:

```
it :: time -> temp          -- inside temperature
ot :: time -> temp          -- outside temperature
dt :: time -> temp          -- difference inside – outside

dt (t) = it (t) – ot (t)
```

The table above (Table 2) can be seen as a function, for each time point we measured a value on the temperature scale. Given that we have only discrete observations, the temperature between observation times must be interpolated(Vckovski 1998). Interpolation is meaningful here, as we know from physics that temperature is a continuous function!

4. INTENSIONAL AND EXTENSIONAL DEFINITION OF FUNCTIONS

Functions are defined as formulae, which permit to compute for any value x a corresponding function value f(x). This is an intensional definition (it is not 'intentional', but one can think that the formula gives the intention of the function). The alternative is an extensional definition: the function is given by a set of values, between which we may interpolate (Figure 169). Interpolation methods must be selected appropriate to the type of process that changes the value(Vckovski 1998; Vckovski and Bucher 1998). The table above (Table 2) gives an extensional definition for inside and outside temperature at a specific location and day.

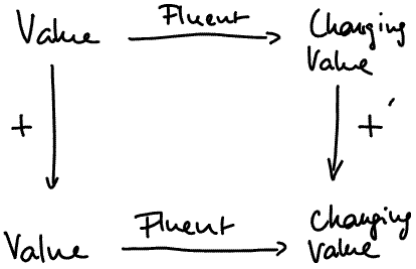


Figure 168: Commutative diagram for fluent

5. THE FUNCTOR FLUENT

Fluent is a functor, a mapping from an ordinary value to a function from time to a value such that that categorical diagram commutes (Figure 168). The functor fluent maps a constant value to a constant function, which returns the same value. A function, like +, is mapped to a synchronous operation on the functions

$(a +' b)\ t = a\ (t) + b\ (t).$

This mapping preserves identity (0 for the operation +) and composition

$a, b \longrightarrow a'(t) = a \quad b'(t) = b$

$a + b \longrightarrow f(t) = a'(t) + b'(t) = a + b$

identity : $0 \longrightarrow z'(t) = 0$

$a + 0 = a \qquad a'(t) + z'(t) = a + 0 = a$

Composition of functions:

$(a + b) + c \longrightarrow \big(a'(t) + b'(t)\big) + c'(t) =$

$(a + b) + c$

The transformation of an operation applicable to a single value to produce a function to work on a fluent is a second order function, which we will call *lift*. Different second order functions are necessary to lift a constant function, a function with one argument, a function with two arguments etc. These will be called *lift0, lift1, lift2* respectively if it is necessary to differentiate them (note that *lift1* is often called *map(Bird 1998)*. A functor must preserve function composition and identity function, i.e.,

lift (a . b) = lift a . lift b,

lift 0 = 0.

Composed functions given by formulae can be lifted by lifting each component of the function. For example the calculation of the percent difference between inside and outside temperature is obtained mechanically by first converting the infix notation in the formula in prefix functions with arguments. For example *a – b* becomes *plus (a, neg (b))*. Then these functions are lifted:

$$d = t_i - t_o$$
$$= \text{minus}\ (t_i, t_o)$$
$$= \text{plus}\ (t_i, (\text{neg}\ t_o))$$
$$d(t) = d\ (t_i(t), t_o(t))$$
$$= \text{lift2}\ \text{plus}\ (t_i(t), \text{lift1}\ \text{neg}\ (t_o(t)))$$

This lifting of functions with a functor is so mechanical that it can be automated; for example the language Haskell (Peterson, Hammond et al. 1997) includes a mechanism that automatically lift functions from working on a single value to a series of values.

Fluents can be constructed from any data type. Güting has proposed a second order operator τ with the same intention(Martin Breunig, Can Türker et al. 2003), but not considered it in the context of category theory as a functor.

```
type Fluent v = Time -> v
```

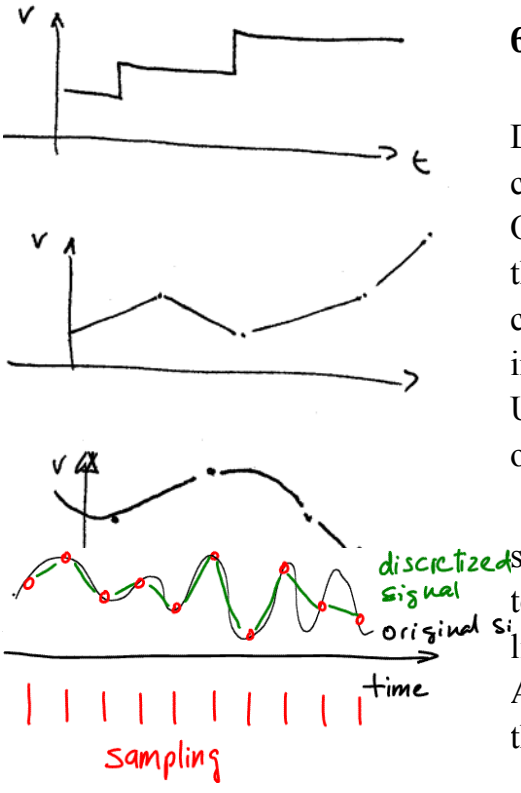


Figure 170: Discretization of signal

6. Discretization of Observations to Obtain a Finite Number of Measurements

Discretization is a form of approximation, namely sampling a continuous signal by a finite number of measurements. Observations must necessarily be for points in time, they *sample* the continuous value, which is called the *signal*. To replace a continuous function with a discrete approximation reduces the information content—something is lost in the discretization. Unfortunately, sampling can also make appear signals that were originally not there (called *aliasing*) (Figure 171).

The sampling (or Nyquist) theorem says: If a signal is sampled with a frequency *f* then the signal must first be filtered to exclude all frequencies higher than *f/2*. If the signal is not limited and high frequencies not excluded, aliasing can occur. Aliasing is the effect that a signal of a low frequency appears in the sampled data where only higher frequencies where present

(Figure 171). Methods to filter are discussed later in this part (see chapter 13).

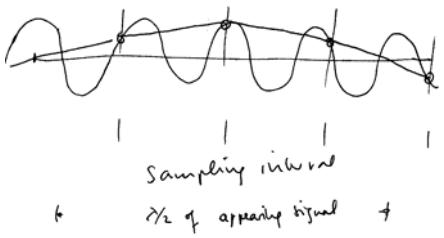


Figure 171: A low frequency signal results from improper sampling of a high frequency signal

The converse is that if we sample a function that is sufficiently smooth no information is lost. If no frequency higher than *f* in the signal occurs, which means no detail smaller than $d=\pi/f$, then sampling with an interval of *d/2* is faithful. Real sensors are not point sensors but have physical extension and integrate over a time interval; this has the same effect than a filter that reduces higher frequencies(Horn 1986 p. 149).

7. TRANSFORMATIONS OF FLUENTS

Imagine that a time series has been observed, for example the temperature in Table 2: Temperature readings inside and outside of a building, and later we determine that the clock used was not set correctly, but was 10 minutes late or was correct at 7:00 but then was running fast, such that it showed 8:00 when it was only 7:55, etc. (Figure 172). Similar transformations may be necessary to change the temperature values if the 0 point and the scale of the thermometer were not correct. These are *1-dimensional* linear transformations.

o = f (t) *-- the original observations*
*k (t) = c + l * t* *-- the correction*
t' (t) = t + k (t)
o' = f (t' (t) *-- o' = o . t'*

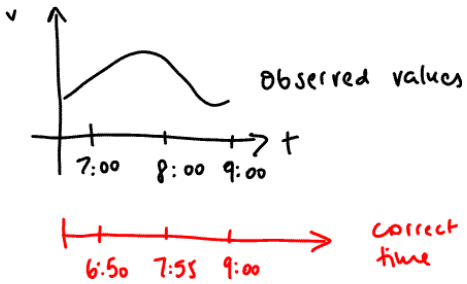


Figure 172: Time correction as a shift and a scale

8. SUMMARY

This chapter has shown how observations of changing values, for example the outdoor temperature during a day, can be seen as functions, in this case a function from day time to temperature. Values changing in time are called *fluents*.

Functions and operations with functions are well-understood in mathematics. Operations defined for a single point in time can be *lifted* to work on time series, combining values synchronously. This systematic lifting is part of the functor, which maps from measurement values to observations in time, which are functions from time to measurement values. The next chapter uses the exact same approach for spatial data.

REVIEW QUESTIONS

- What is a fluent?
- Why is fluent a functor?
- Give an example how to use a synchronous operation.

- What is the sampling theorem? What does it say?
- What is meant by aliasing?

Chapter 12 MAP LAYERS

In this chapter we focus on space and observations of properties in space, which result in measurement values related to locations in space. Sensors, for example areal photographs and remote sensing data captured from satellites produce such measurements. This chapter focuses on the processing of representations of properties of locations in images.

Figure 173: A remote sensing image

Space is continuous and we can observe properties at any location at any time. In this chapter, we focus on snapshots with time fixed. This is sometimes called the *field view*. It answers questions like "what is here?"; the alternative *object view* answers to the question "where is this object" (Couclelis 1992) and will be treated in the next part.

A (x, t) = f (x,y,z, t)
Goodchild's geographic reality (Goodchild 1990; Goodchild 1992)

S (x) = f' (x,y,z)
a snapshot of space, time fixed

Map layers are used, for example, to find an area that is suitable for building a new home, given data sets describing exposition, zoning and current land prices (Figure 174). The focus is on *homological* operations that are the most often used operations from Dana Tomlin's map algebra(Tomlin 1991; Tomlin 1994); other operations are discussed in the next 2 chapters.

Figure 174: Three data sets to help identify an area where I want to build my new home

1. INTRODUCTION

Space is continuous and varies continuously. Our observations of properties at points are related to points in 2 or 3 spatial dimensions and in 1 temporal dimension. The discussion in this chapter is restricted to snapshots with time fixed and *2d* space, which means a projection from the real 3 dimensional world to the projection plane. The remote sensing images from space are good examples. They are raster images, where values are recorded for a regular grid dividing space in equal cells, but the principles discussed here apply equally to irregular subdivision of space (Tomlin 1991; Eastman 1993) but applies as well to other representations (see later chapter 30).

The extension from *2* to 3 dimensions and the combination with the time varying values discussed in the previous chapter are trivial. The limitation to *2* dimensions in this chapter is didactic and does not limit the generality of the results.

2. TOMLIN'S MAP ALGEBRA

One of the original ideas that lead to the development of GIS was the manual map overlay procedures used by planners and geographers for a long time(McHarg 1969; McHarg 1992). Maps are drawn on translucent paper and overlaid on a light table. Visual interpretation allows then to find solution of questions like "where is a south-exposed area zoned residential and reasonably priced" or "find the area where logging of pine trees is permitted, avoiding areas closer than 100 m to a water body". Dana Tomlin, then a student of Joseph K. Berry at Yale University, saw in the late 1970s that such questions can be computerized. It is possible to express them as an algebra of operations on raster. This algebra is closed: the result of one operation is again a map layer and can be used as an input in the next one(Tomlin 1983). He defined Cartographic Model and Map Layer as follows:

2.1 CARTOGRAPHIC MODEL

A cartographic model is a 'collection of maps that are organized such that each of these layers of information pertains to a common site'(Tomlin 1983, 4). The elements of the cartographic model are the layers and these are already 'registered', which means that they cover the same area, have the same orientation

etc. (Figure 175). Linear transformations and map transformations may be used to achieve this registration.

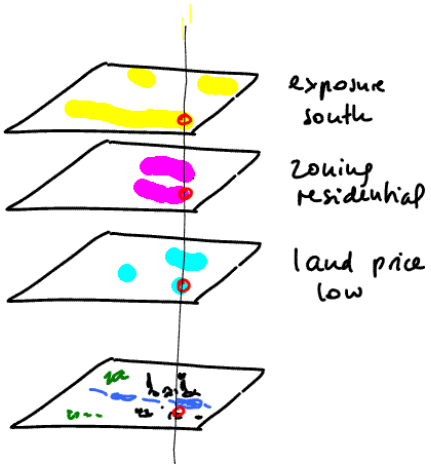


Figure 175: Coordinated layers are combined homologically

2.2 MAP LAYER

The notion map or thematic layer is used to describe a description of one property with respect to its spatial distribution. A map layer is one theme from a cartographic model, it is 'more like a map of just one of an area's characteristics'(Tomlin 1983, 6).

The metaphor *layer* is used because a GIS is sometimes seen as a 'layered cake' of thematic layers, which are stacked one above the other (for a discussion of effects of this metaphor see(Frank and Campari 1993). Molenaar has used the term *'single value map'* to differentiate it from maps that contain more than one variable(Molenaar 1995; Molenaar 1998). This is an unnecessary differentiation; technically they are single values, namely tuples consisting of several values.

We will use the word *field* for the concept of continuous space and *raster* for the square grid discretization of it. This use of '*field'* should not be confused with the algebraic structure field, encountered in chapter 5.

Field = continuous space
Raster= a regular (square) discretization of a field

2.3 OPERATIONS ON MAP LAYERS

Map layers can be 'overlaid' and areas where some combination of values from one and the other layer occur identified. Planners and cartographers used to trace such areas on a new sheet laid on top of the pile (Figure 176). The overlay shows where the three properties apply and this can be traced on a new sheet(McHarg 1969; McHarg 1992). This new sheet can then be used in another overlay operation, meaning that these operations form a closed algebra.

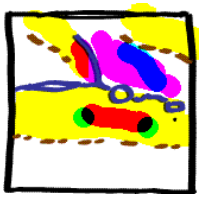


Figure 176: Overlay of the three layers of Figure 174: Three data sets to help identify an area where I want to build my new home

The manual operations on a light table limit the number of layers that can be combined and the tracing of new layers is a time consuming operation. Photographic processes were occasionally used, but give little additional flexibility. Only the computerization opened the door for a flexible combination with more operations than the manual overlay.

2.4 CLASSIFICATION OF OPERATIONS

Tomlin differentiated operations in map algebra into three groups:

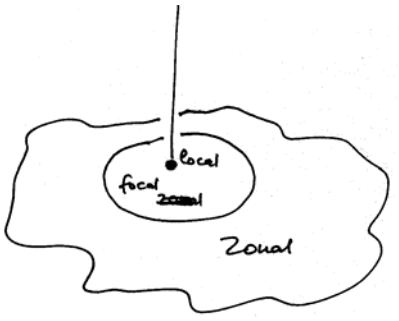


Figure 177 Local, focal, and zonal operations: area of support to compute a single new value

- Local operations, which combine the value from the same location
- Focal operations, which combine values around a focal point to a single value
- Zonal operations, which combine values from a single zone.

A local operation is looking at a single point and the resulting value is the combination of values from this point; focal operations consider the area around the point of interest; and zonal operations consider values from an area within an irregularly formed zone (Figure 177). Tomlin's definition of a zone is an area where the same value obtains, but not necessarily connected. The three layers in Figure 174 are showing each a zone before a background of *null* values.

A zone can be defined as a geographic area exhibiting some particular quality that distinguishes it from other geographic areas.(Tomlin 1983 p.10).

Tomlin's book gives a wonderful collection of functions that can be used to transform and combine layers that are meaningful in a planning application(Tomlin 1990). In this part, his ideas are reviewed from a mathematical (categorical) point of view.

Homological means at the same location.

3. Local Operations Are Homologically Applied Operations

Homological operations combine values from one or several layers for one location at a time. They cut the values from each input layer at the same location and produce from this set of values a single result, which is the value in the result layer. The vertical "pin" in Figure 175 indicates this connection between *homologous* values.

Local operations are given as functions that take one or several values as inputs and compute a single value as a result. For example, given the function *g* to compute a new value from values *a, b,* and *c* from Layers *A, B,* and *C* is $c(x,y) = g\,(a(x,y), b(x,y))$. We combine the values for corresponding (homological—same location) points with the given function.

The number of values that are combined is arbitrary; functions that take one layer and transform it into another layer are called *classifier* or *reclassifier.* These and functions that take two values and combine in a single new value are the most used ones; functions with more values occur occasionally. A simple example: given the two layers of male and female average population per areal unit, we need to compute the average population per areal unit. This is simple addition of the value for each location. The result can then be classified for areas with an

average population higher than 5 (Figure 178). This is similar to adding two time series (previous chapter).

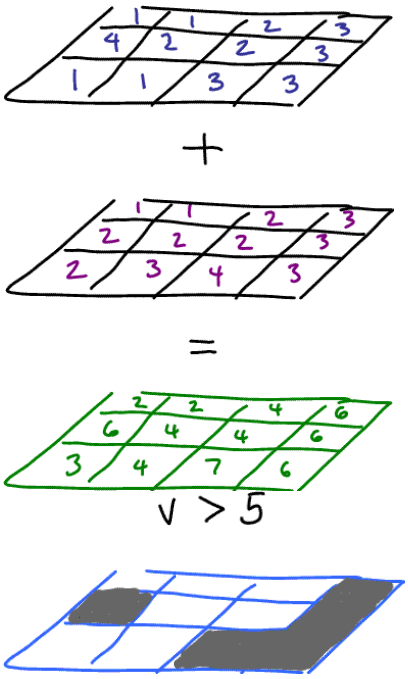


Figure 178: Adding to layers and then reclassify the result

Combine layers m and n to give layer l with formula l = f (m, n)
l(x) = f (m(x), n(x))

4. MAP LAYERS ARE FUNCTIONS

Map layers can be seen as functions from a location to a value. Observations will be available only for specific points and other locations must be interpolated(Vckovski 1998).

layer: location -> value

Operations on layers are defined as homological application of the function to each location in the layer. If the values *a* and *b* are available for each point in space, i.e., if they are functions *a (x,y)* and *b(x,y)* and we are interested in the values *v = g (a,b),* then we can construct a function *v (x,y) = g (a(x,y), b(x,y)).*

Tomlin's description of Map Algebra is not typed and most implementations today do not apply a type concept to the map overlay operations. The combination of layers can be checked for correct types: the operation must apply to a layer or layers that produce the correct types for the operation; then the resulting layer is a function from location to the result type of the operation. The types for the above operation g combining two layers must be:

```
g :: type1 -> type2 -> resultType
a:: location -> type1
b::: location -> type2
v :: location -> resultType.
```

5. THE FUNCTORS LAYER

The map *layer* is a functor; it converts operations on single values to operations on a function from a location to a value. It is similar to the construction of *fluent*, which is also a functor (see previous chapter).

Understanding that layers are functors gives us access to the same second order function *lift* used to combine time series (chapter 11). *Lift* takes a function on values and produces functions on layers of values. Any operation with the right type can be used to transform a layer or to combine layers. Often used operations are:

- Classification: the values in a layer are classified according to some criteria; such operations transform a layer with floating point values to a layer with ordinal or nominal values.
- Boolean operations used to combine layers and find areas where two attributes apply (*AND*) (Figure 179) or where either of two attributes applies (*OR*).

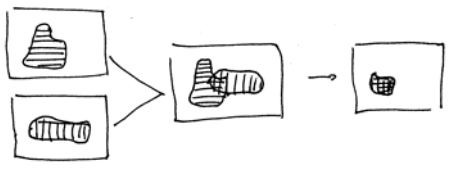


Figure 179: Boolean AND gives intersection

- Arithmetic operations on values: *+, -, *, /.*
- Operations using order: *min, max*.
- Statistical operations: *sum, average, median*, most frequent value.
- Other functions: *square root, sine, cosine, tangent, arc sine, arc cosine, arc tangent(Tomlin 1990, 65)*.

Homological operations are used in combinations: firstly, values are changed through some formula and then a classification is applied and last, the result then combined with some other layer. Because *layer* is a functor, composition of functions is mapped correctly and it is possible to simplify such operations using the formulae about distribution of *lift1*:

(lift1 f . lift1 g) l = lift1 (f.g) l

If the functor *layer* is defined, then all operations on single values can be lifted and used to combine layers as they would combine single values. This can be used to construct new formulae or new rating methods and lift them to apply to layers.

For some operations, specially constructed layers are useful. For example, a layer that contains the coordinates of the points (or its discrete equivalent), is a function *id* (*f (x,y) = (x,y)*). With such layers, it is, for example, possible to calculate the distance from a given point *p,* lifting the function *dist (p,_)*.

6. Map Layers Are Extensionally Defined Functions

Layers that are defined intensionally as functions and given as a formula are seldom in geography. Most often, values are recorded for points and interpolated between them or regions where the same value obtained are identified. Other representations are possible for continuous functions. For example, Waldo Tobler has computed the coefficients for an approximation of the population density of the world using a series of spherical harmonics(Tobler 1992). This gives for world population an intensionally defined function.

The restrictions on discretization discussed in the chapter on fluents (chapter 11), applies in *2* or *3* dimensions as well. It may be surprising to think of frequencies in space, but it opens the conceptual framework of signal processing (Horn 1986) for application to geography, which will be explored extensively when discussing approximations.

REVIEW QUESTIONS

- What is meant by the expression 'homological operations'? Give an example.
- Define layer.
- What is the notion 'field' meaning here? What other meanings do you know?
- What are local, focal, and zonal operations? How are they differentiated?
- In what sense is map algebra closed?
- How would you calculate in a discrete raster representation a layer that contains the distance to a given point *p*?
- Give a local function that classifies a layer with the height in meters, such that areas below zero, between zero and 500, 500 and 1000, etc. are separated.
- Give a small part of a layer as an extensional definition.
- How does the sampling theorem apply to space?

Chapter 13 CONVOLUTION: FOCAL OPERATIONS FOR FLUENTS AND LAYERS

Map algebra does not only contain operations that work on a single location using data from one or more layers, but includes a number of methods to work on neighborhoods around a location. *Focal* operations apply to a point and the immediate neighbors around it. Similar operations are known in image processing for smoothing time series and images. They are called *convolutions* (German term: Faltung).

In this chapter the concept of focal operations is first explained for time series (*fluents*) because the explanations are simpler and then applied to *layers*. The chapter first concentrates on the convolution operation that is defined for continuous functions and generalized this concept in the last part of the chapter to other focal operations that are not immediately expressed as convolutions. Convolution operations have a fixed size of the kernel and introduce a scale dependency in their result; an extensive discussion will later relate them to error treatment.

Figure 180: Rate of influence decreases with distance from focal point

1. INTRODUCTION

A large class of interesting operations on layers computes a new value using all the values in the field but using a weighting function to give more influence to the neighbors than to locations further away. According to Waldo Tobler's first law of geography, in most cases influences of far away things are negligible and we can restrict the area of influence to a small neighborhood around the point of interest, the *focal point* (Figure 180). This general principle is powerful and has wide applicability; it is used in signal processing and remote sensing to smooth or enhance images, and it can also be used in GIS.

First law of geography:
All things influence all other things;
nearby things influence more.
Waldo Tobler

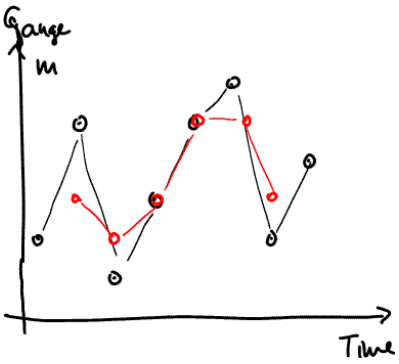


Figure 181: Original and smoothed values

Time	Value	Smooth Value
10:01	12	
02	15	13
03	11	12
04	13	13
05	15	15
06	16	15
07	12	13
08	14	
.		
.		

Figure 182: Sliding average

2. CONVOLUTION FOR FLUENTS

2.1 EXAMPLE: SLIDING AVERAGE

Consider the practical problem of measurements in a time series; for example the water height at a water gauge station in a lake. Waves, random errors and noise may produce rapidly varying readings when we know that the water level varies only slowly. A *sliding average* to smooth the time series is routinely used; this is computed with a formula that takes *1/4* of the value before, *1/4* of the value after and *1/2* of the current value—and this formula is applied to every value in the series (Figure 182). The smoothing effect of the sliding average is visible (Figure 181)!

$$s_n = \frac{v_{n-1} + 2 \times v_n + v_{n+1}}{4}$$

2.2 CONVOLUTION FOR CONTINUOUS FUNCTIONS

Convolution is defined as the integral of the product of two functions; one is the signal *f(t),* the measured value, the other the weighting function *h(ξ),* which determines how much influence the values have. The result at the point *t* is the integral of the product of these two functions:

$$g(t) = \int_{-\infty}^{\infty} f(t-\xi)\, h(\xi)\, d\xi$$

Convolution is commutative and associative. The sliding average worked with discrete values and used a weighting function that is 0 everywhere except *1/4* for -1 and 1 and *1/2* for 0. The following computation shows how convolution works as a multiplication of two functions given as polynomials (Figure 183):

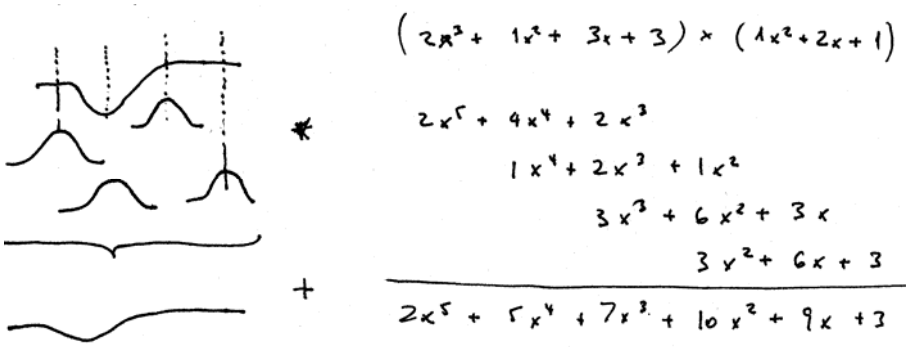


Figure 183: Convolution of functions given as polynoms

2.3 CONVOLUTION IS LINEAR AND SHIFT INVARIANT

The convolution operation has two properties that are important for temporal and spatial problems: it is *linear*, which means that twice the input gives twice the output:

input: $\alpha f_1 + \beta f_2$
output: $\alpha g_1 + \beta g_2$

Functions which are linear are independent of the units used for measurements.

It is *shift-invariant*, which means that it is invariant under shifting of the coordinate system:

input: $f(t-k)$
output: $g(t-k)$

It can be shown that all linear and shift-invariant transformations can be described as convolution with some function *h(Horn 1986, 109)*.

2.4 CONVOLUTION FOR SERIES WITH DISCRETE VALUES

Convolutions can be discretized. The fluent is given by a sequence of equidistant values $v_1, v_2, \ldots v_n$ and the weighting function is given by a *stencil* (sometimes called the convolution kernel) $w_1, w_2, .. w_m$. The stencil to compute the sliding average before was (*1/4, 1/2, 1/4*). The length of the stencil indicates the size of the neighborhood that influences the result; usually the stencils are small, three or five values are usually sufficient. The computation consists of sliding the stencil along the time series and multiplying the values with the corresponding weight in the stencil and to sum these products (Figure 182).

The discrete form of convolution appeals to intuition and is easy to compute and visualize. It contrasts in this respect sharply with the abstract definition of convolution as an integral of the multiplication of two functions.

2.5 CONVOLUTION FOR SMOOTHING A FLUENT

The best weighting function to smooth a fluent is a Gaussian function (Figure 184):

$$h(t) = \frac{1}{2\pi\sigma^2} e^{-\frac{1 \cdot t^2}{2 \cdot \sigma^2}}$$

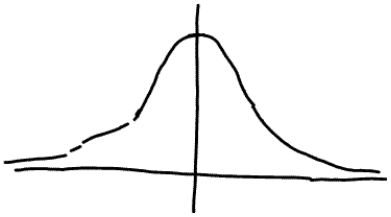


Figure 184: A Gaussian function

The stencil in Figure 182 is a length three discrete form of a Gaussian: *1/4, 1/2, 1/4* or *1/4 (1, 2, 1)*.

2.6 CONVOLUTION TO DETECT EDGES

The derivation of a signal function accentuates the edges. The derivation of a signal can be computed as a convolution, because derivation of a function is linear and shift-invariant. To identify

the function that is the equivalent convolution to the derivation is not simple, given that derivation is not an ordinary function *f(x)*. Intuitively, the values for the weighting functions must give a high value to the center and negative values to the neighbors (Figure 185). For discrete values, the stencil is for example: (*1/2, -1, 1/2*).

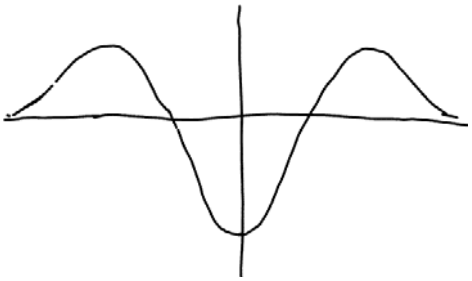


Figure 185: Laplace operator used as an edge detecting convolution (Mexican hat)

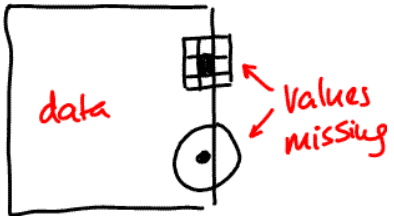


Figure 186: Values missing to compute convolution along an edge

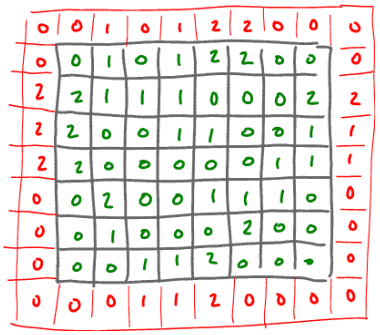


Figure 187: A layer with additional values around its boundary to permit convolution with a 3 by 3 stencil

2.7 Treatment of time series with not equidistant values

Understanding convolution as the multiplication of two functions permits the generalization of the operation to time series where the values are not equidistant. The weighting function *h* is a function of the distance to the focal point *x* as was shown in Figure 183.

3. Problem with Edges

The values at the points along the edge of the area treated are computed from values around this point—but part of this neighborhood is not available (Figure 186). This is a practical problem for computation. Several solutions are possible:

- The area of interest is inserted in an infinite area with value 0. This allows computation up to the edge, but values near the edges are not correct.
- For the area around the edge where not enough data is available, no values are computed. This produces correct values, but the area covered becomes smaller with each convolution.
- The image is mirrored at the borders (Figure 187); this gives not correct values for the cells along the edge, but the values are at close.

4. Convolution in *2D* for Layers: Focal Operations within Neighborhoods

Convolution can be extended from one dimension to multiple dimensions. It is used to process images, including remote sensing images, and geographic data processing. Convolution in *2*-dimensions is generally useful for spatial analysis, to smooth a surface or to detect edges, etc.

Convolution for layers in *2*-dimensions is defined like convolutions in *1*-dimension, except that both the signal and the weighting function are in *2*-dimensions.

$$G(x,y) = \iint_{\xi\,\eta} F(x-\xi, y-\eta) * H(\xi,\eta)\, d\xi\, d\eta$$

Convolution for layers is linear and shift-invariant. Shift-invariance for images has intuitive meaning: an image of an object taken from a position and the image taken from a slightly shifted position should be very similar (Figure 188)!

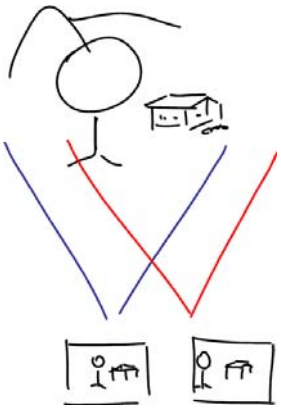


Figure 188: A picture and a second one from a shifted position

A transformation in 2 variables is bi-linear, if a linear transformation of any or both of the inputs produces a linear transformation of the result. A transformation is shift-invariant if it produces the shifted output *g (x-a, y-b)* when given the shifted input *f (x-a, y-b)(Horn 1986 p. 105)*; this is verified by inserting in the formula above.

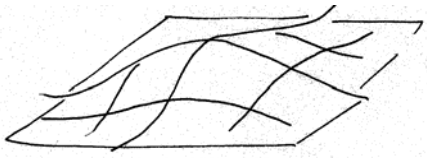


Figure 189: An example for H (Gaussian)

4.1 SMOOTHING OF A LAYER

Convolutions can be used to filter an image to exclude high frequencies (detail); a Gaussian filter that attenuates higher frequencies is often preferred over a sharp low pass filter, that cuts frequencies at a precisely defined limit(Horn 1986, 127).

A convolution to smooth a layer uses a Gaussian function in *2*-dimensions (Figure 189). The effect is the same as we found for smoothing time series (Figure 182), applied to *2* dimensions. The Gaussian function is rotationally symmetric and its effect in the convolution is also rotation invariant. This means that the convolution of an image is the rotated convolution of a rotated image: *R (conv a b) = conv (R a) b.* A stencil with discrete values for this function is given in Figure 190.

$$h(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{1}{2}\frac{x^2+y^2}{\sigma^2}}$$

$\frac{1}{16}$

1	4	1
4	-20	4
1	4	1

Figure 190: Convolution Stencil for a Gaussian

4.2 EDGE DETECTION IN LAYERS

For the detection of edges, another rotational symmetric function, namely the Laplacian operator can be used:

$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

Given that any shift-invariant and linear system is a convolution, such a function must exist. Horn gives a function that is the limit of a sequence of functions(Horn 1986, 122):

$$L_\sigma(x,y) = \left(\frac{x^2+y^2-\sigma^2}{2\pi\sigma^6}\right) e^{-\frac{1}{2}\frac{x^2+y^2}{\sigma^2}}$$

From this we can deduce a piece-wise constant function, which then leads to a stencil (Figure 191):

$$L_\varepsilon(x,y) = \begin{cases} -2/(4\pi\varepsilon^4) & \text{for } 0 \le x^2+y^2 \le \varepsilon^2 \\ +2/(3\pi\varepsilon^4) & \text{for } \varepsilon^2 \le x^2+y^2 \le 4\varepsilon^2 \\ 0 & \text{for } 4\varepsilon^2 < x^2+y^2 \end{cases}$$

4.3 *Isotropic and non-isotropic convolutions*

Convolution of multi-dimensional fields is isotropic, treating space in all directions the same, if the functions used for the convolution is rotationally symmetric, i.e., invariant under rotation: *R (f a) = f (R a)* where *R* is a rotation matrix. The Gaussian and Laplacian convolutions are examples for rotationally symmetric convolutions. Convolutions applied to raster require symmetry only for the directions where the raster itself has symmetry; for square rasters, these are quarter turns.

Convolutions can be anisotropic. Most are detection of edges in a specific direction (Figure 192). For this a function is used that is not rotationally symmetric nor are the stencils.

$\frac{1}{16}$

1	2	1
2	4	2
1	2	1

Figure 191: A stencil to detect edges (Laplacian)

1	-2	1
1	-2	1
1	-2	1

Figure 192: An anisotropic stencil to detect edges in north-south direction

$$f(x,y) = \begin{cases} 1 & x^2+y^2 < c \\ 0 & \end{cases}$$

Figure 193: Function constant in an interval

1	0	0
0	0	0
0	0	0

Figure 194: Convolution that is a shift right and down by one cell

5. Other Focal Operations

Many of the focal functions described by Tomlin (Tomlin 1990) can be constructed as convolutions. For example, the *local sum* operator is a convolution with a function that is constant for some distance (Figure 193). The focal average is the result of *local sum* divided by the area in the convolution function, which is $\pi * c^2$. The corresponding stencils are easy to derive.

Functions useful for applications are composed from a focal operation and other, local operations. Any combination of values in the stencil can be computed in 2 steps: first apply stencils, which have only a single 1 in one of the cells: this is a shift operation for the layer (Figure 194). Then these shifted layers are combined with local operations.

Some focal operations provided in GIS include the central value (v_{22} in Figure 195), some do not (and sometimes it is unclear, if it is included or not). Many useful functions are essentially statistics of the 9 (or 8) values cut out by the stencil. This includes *focal maximum* and *focal minimum*, which apply a *max* or *min* function to the values returned, *focal sum* and *focal average*, but also *focal variety*, *focal majority*, *focal minority*, *focal median*.

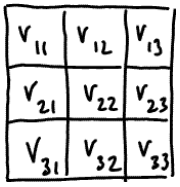


Figure 195: The values cut out by the stencil

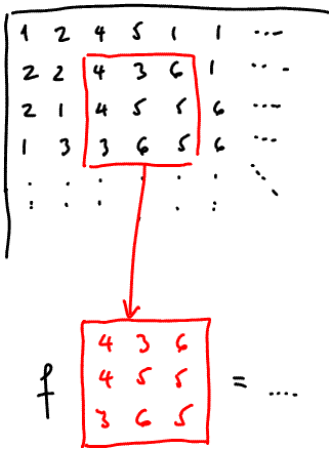


Figure 196: An example layer and the computation of a new value

Other focal functions compare the environment with the central value (v_{22}). For example, local percentile gives the percentage in the environment (v_{11}... v_{33}, but not v_{22}) that is less than v_{22}. Tomlin shows how this function can be used to compute how prominent a place is from the height data, depending how much of its environment is at a lower altitude.

With focal operations it is possible to compute the *gradient* (terrain inclination, angle of the terrain with the horizontal direction) and the *aspect* (direction of the maximum gradient). It is also possible to determine the direction of water-flow over the area, which may lead to a determination of streamlines(Frank, Palmer et al. 1986).

The cost of traveling from a given point over an anisotropic surface, i.e., a surface where travel cost are different for each point, given as a layer *travelCost:: x -> c*, is computed as a fixed point of a convolution. Regular convolution operations calculate a new layer from a given one in a single sweep; the new values derive from the given ones only. To compute the traveling cost requires spreading the cost of reaching a location from the starting point over the surface; it is a repeated application of a convolution till a fixed point $f(x_{i+1}) = f(x_i)$ is reached.

There is a tradeoff between offering a large number of specialized operations and providing only a small set of building blocks from which others can be constructed. I prefer the building blocks because the effort to understand what the given prefabricated operations do requires often more effort than constructing new ones; and one finds at the end, that the given function is not what is required to solve a problem.

6. CONCLUSIONS

The focal operations are defined without reference to a representation. Convolution is explained in terms of continuous functions and applies to any form in which a function *f: (x,y) -> v* can be represented. The implementation as operations on arrays for the discrete case – regular raster (Figure 197) – is offering itself, but it is just a convenient implementation (some of the specialized focal operations suggested by Tomlin seem to assume a raster representation).

Continuous functions avoid the problems of discretization and guarantee that the results are not dependent on the resolution selected (but still depend on the size diameter of the kernel

function). Understanding focal operations as convolution leads towards the generalization of focal operations from raster to other irregular tessellations (Figure 198).

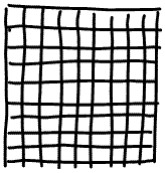


Figure 197: A regular subdivision

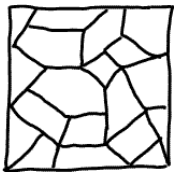


Figure 198: Irregular subdivision

A method to compute convolutions for subdivisions represented as irregular tessellations is to convert the integral into a finite sum and to sample the layer at the appropriate points; this is essentially a conversion of the subdivision in a raster representation of a suitable resolution, which can be achieved without storing the raster.

In signal processing a different approach is often selected: transform the signal from the temporal (or spatial) dimension into the frequency domain by a Fourier transformation. A convolution in the time domain is a multiplication of functions in the frequency domain. This can take advantage of the Fast Fourier Transformation algorithm. It would be useful to explore its usefulness for geographic data processing.

A convolution is a multiplication in the frequency domain.

I feel that a careful analysis of the functions required to solve practical problems is warranted and expect that some generality is found. Tomlin also includes visibility based on line of sight from a given point as a focal operation and an operation to identify connected areas with the same values. These two functions seem not to fit within the framework of convolution and need further study.

REVIEW QUESTIONS

- Demonstrate by computing the linearity and shift-invariance for (*1*-dimensional) convolution.
- Detect the time when temperature dropped or increase most in the time series: 10, 11, 12, 15, 16, 16, 17, 12, 10. What operation are you using? What is the stencil?
- Give a stencil for a *2d* local average.
- Why is it that convolution (and other functions in GIS) are linear?

Chapter 14 ZONAL OPERATIONS USING A LOCATION FUNCTION

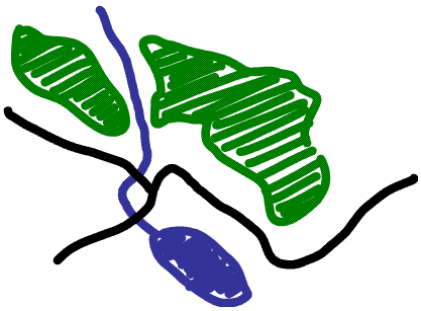


Figure 199: A land use map with wood, water and street use

We can focus our attention in a layer on all areas that have the same value: we can look at all the wooded or all the urbanized area in a map, we can look at lakes, etc. (Figure 201). Tomlin calls such a selection of areas with the same value a *zone* and Map Algebra contains a number of functions operating on zones. Zonal operations compute a new value for a location based on all areas that has the same value as this one; zonal operations combine in a geometrically varying way a location with other similar locations.

Zonal operations are used to compute the area of zones, for example, how much area is forest in Figure 199? They can be used to determine the center of gravity of a zone, the average distance of the zone from a given point, the average height above sea level of a zone, etc.

Definition:
Zone = Area with the same value.

Zones are an intermediate step towards the focus on objects, which is the second half of this book. It is to differentiate zones from objects—zones are all areas with a value, they are a layer; the zone wooded area is different from the two objects "wood" in Figure 201.

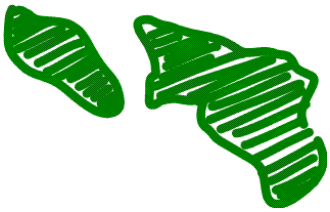


Figure 200: The zone 'wood'

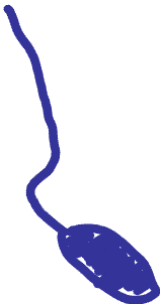


Figure 201d: The water zone

1. DEFINITION OF ZONES

Zones are defined as all areas that have the same value. This is immediately meaningful for layers that are functions from space to discrete values (e.g., integer, nominal). For layers that map to continuous values (e.g., real numbers), it is usually necessary to classify the layer first into a small number of classes.

Image processing thresholds images obtains images with Boolean values that are called binary images; these are similar to zones. The image as a function is then called a *'characteristic function'(Horn 1986, 47)*.

2. CLOSEDNESS OF ZONAL OPERATIONS

Tomlin assigns to each location a value from the zonal operation. This is different from operations in image processing, where functions applied to a zone (a characteristic function) produce a single value. The result of a zonal operation is the same for all

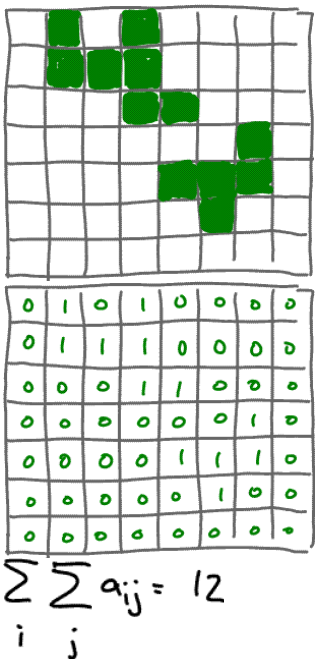


Figure 202: Aggregate the values in the zone

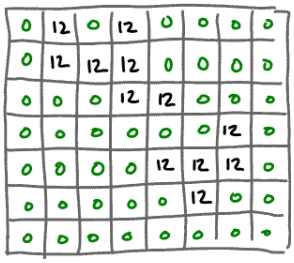


Figure 203: The result of the zonal area operation

Values in a set are always different!

locations in the zone and is filled in all cells of the zone. For example at the end of a zonal operation "area of zone" all cells of a zone contain the value for the total area of the zone.

This duplication of values is necessary to make zonal operations produce a layer and assures that map algebra is closed.

3. Computational Schema of Zonal Operations

Zonal operations are combinations of local operations and a new '*integrate over layer'* operation. Take a simple example, namely the computation of the area of the 2 zones forest and water of Figure 201:

Assume a classified layer *M :: x -> {f, w}*.

1. Create two Boolean functions *lf, lw :: x -> Bool*, where *true* means that the location is in the zone and *false* outside.

2. Classify the layer M with these two functions *lf, lw*; this gives two layers :: *x -> {0, 1}* (characteristic functions).

3. Integrate over layer: aggregate all values in each layer to compute a value v for the zones v_f and v_w.

$$\text{Area} = \int_{\text{layer}} f(x)\,dx$$

4. classify the zones in the layer *M* with a map *f -> v_f, w -> v_w, s -> v_s, l -> v_l*; this fills back the computed values in all cells of the zone.

This is not a description of an implementation but it describes the logic of zonal operations. Special operations differ in the function *f* which is used and the function to aggregate the values across the area.

Note that the operation 'integrate over layer' is independent of the representation. For a raster representation, the integral becomes a sum (Figure 202), in the simplest case just a count. For other representations, a method to sum a function over a layer must be given.

4. Number of Zones in a Layer

The number of zones in a layer is less than or equal to the cardinality of the set of values of the layer. In the limiting case, each location is a zone by itself, but then zonal operations are the same as local operations.

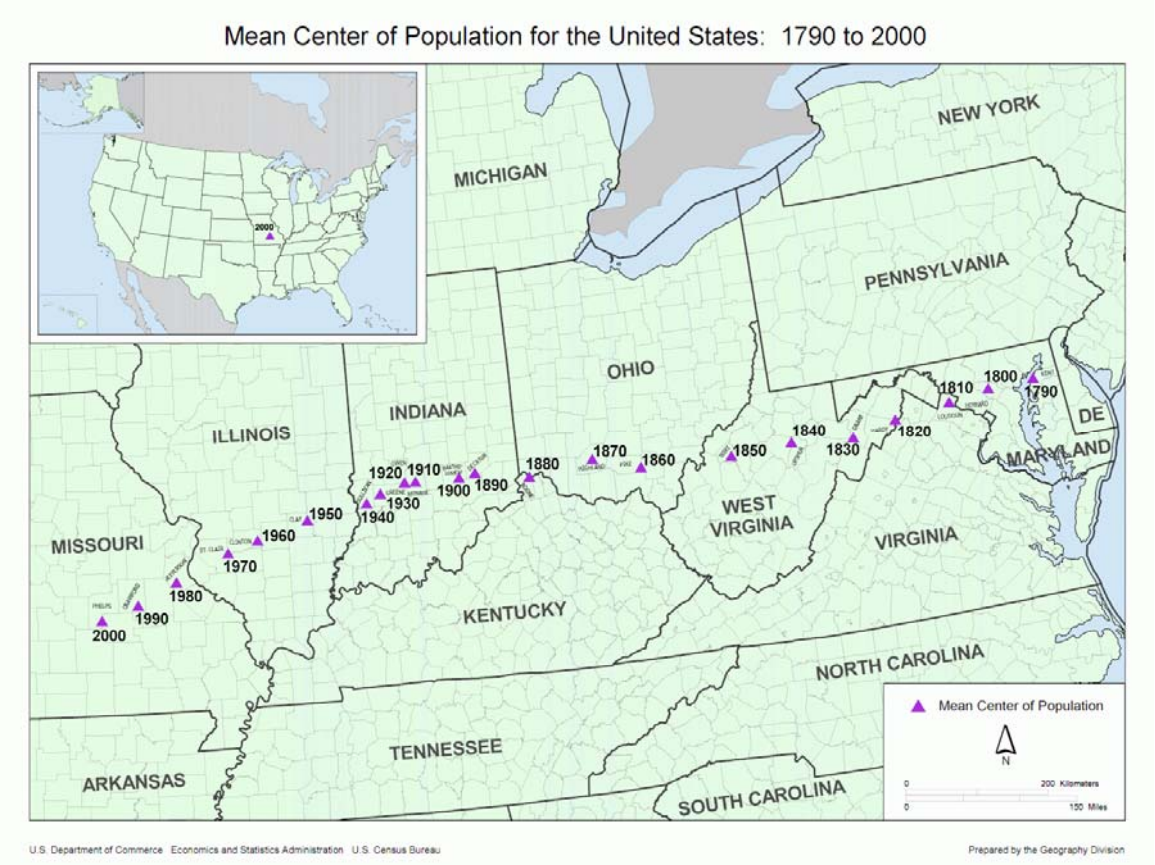


Figure 204: Mean Center of population of USA (source: http://upload.wikimedia.org/wikipedia/en/2/27/Mean_ctr_pop_US_1790-2000.png

5. Zonal Operations with Meaningful Second Layer

Some of the operations on a zone use a second layer, a layer different than the layer used to form the zone. The zonal operation then obtains values from this layer that are combined to give the value for the zone. For example, one may ask "what is the average height of the forested land". Forest land means a zone, formed on land use, but the average is then for the height, which comes from a second layer.

Operations using a second layer can compute arbitrary functions that combine the values of this second layer in the zone in a single value. It is not necessarily the function 'integrate over layer' with addition, but the + operation in the aggregation can be replace by other operations. Functions that are used are *sum, max, min, mean, product, variety, majority*, etc.

Tomlin introduces Partial zonal operations: The values in a zone can be compared with the value at the given location. For example one asks for a point in a zone, how much area of the zone is higher than the given point. Such operations can be composed from the basic operations and I doubt, that it is worthwhile to include such special operations into a GIS for the few cases they are used.

6. CENTROID AND OTHER MOMENTS

The center of gravity is a geographically meaningful concept. It is instructive to draw, for example, the movement of the center of gravity for the population of the USA over the past 200 years. The movement of the center of gravity for the population shows in a nutshell the movement first towards the west and later in the 20th century to the south (Florida, Arizona).

In statistics, the centroid is just a *moment*, it is the first moment divided by the area (which can be seen as the zeroth moment), and the second moment is the standard deviation. The second moment has a physical interpretation as the inertia against rotation around an axis. This can be used to determine the orientation of the object as the axis which has least (or maximum) inertia(Horn 1986). Higher moments can be constructed but are seldom meaningful.

Moments are additive

Moments are characteristics of a zone that are additive. If two disjoint zones are combined to form a single one, the moments add: $M\ a + M\ b = M\ (a + b)$. This means, that the moment of a collection of objects and the moment of the composed object is the same. For computation it means, that moments can be computed for parts and the results for the parts combined. The center of gravity of a figure is the center of gravity of the parts, each part represented by its center of gravity multiplied with its mass.

Disjoint = no common part

6.1 CENTROID

The centroid is the center of mass of an object. It is computed as the first moment of the object divided by the area, because the moment—physically the force to turn the object around this point—must be zero for the centroid. The same computation applies to zones.

We sum the contribution of each part of the object to turn around the origin in direction of negative y (respective negative x) (Figure 205). This must be equal to the total area (mass) of the object times the distance of the centroid from the origin

$\overline{X} * m0 = m1x.$

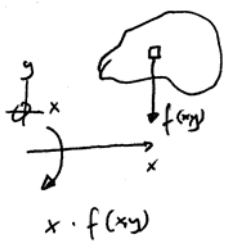


Figure 205: Sum the contribution of each element in the object

The coordinates of the center of gravity is the first moments divided by the zeroth moment (Figure 206).

$\overline{X} = (m_{1x} / m_0, m_{1y}/m_0)$

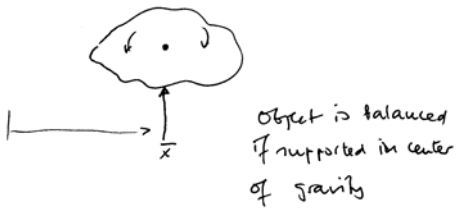


Figure 206: Object in equilibrium

To calculate the center of gravity a second layer which gives the first or second coordinate is necessary. The formulae for these layers, which are used for all calculation of moments, are

$f(x,y) = x$
$f(x,y) = y.$

6.2 HIGHER MOMENTS

$$\text{Moments}: \quad m_0 = \iint f(x,y)\,dx\,dy$$

$$m_{1x} = \iint x \cdot f(x,y)\,dx\,dy$$
$$m_{1y} = \iint y \cdot f(x,y)\,dx\,dy$$

$$m_{2a} = \iint (x-\bar{x})\, f(x,y)\,dx\,dy$$
$$m_{2b} = 2 \iint (x-\bar{x})(y-\bar{y})\, f(x,y)\,dx\,dy$$
$$m_{2c} = \iint (y-\bar{y})\, f(x,y)\,dx\,dy$$

$$\text{center of mass}: \quad \bar{x} = m_{1x}/m_0, \quad \bar{y} = m_{1y}/m_0$$

$$\text{orientation} = \quad \sin 2\alpha = \pm \frac{m_{2b}}{\sqrt{m_{2b}^2 + (m_{2a} - m_{2c})^2}}$$

6.3 ORIENTATION OF THE AXIS

The axis of an object can be found as the direction for which the second moments are minimal, i.e., the axis around which the object is easiest to turn. To find the axis, the integral

$\int r^2 f(xy)\,dx\,dy$

must be found, where r is the distance of any point to the axis. Expressing r as a function of the axis as Normal Form

$x \sin\alpha - y \cos\alpha + c = 0$

and integration leads to the solution of a quadratic equation in $\sin 2\alpha$.

$$ax^2 + bx + c = 0$$
$$x_{1,2} = -\frac{b \pm \sqrt{b^2 - 4ac}}{2a}$$

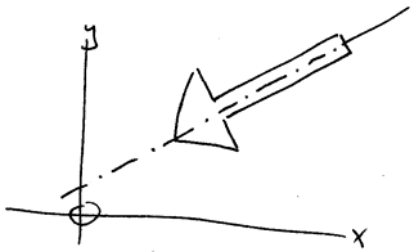


Figure 207: Axis of an object

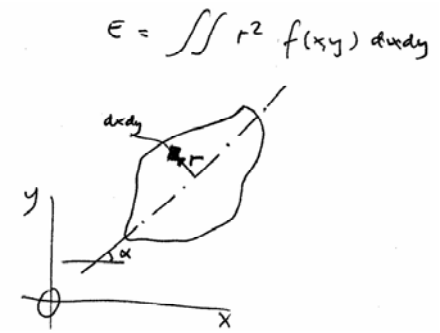


Figure 208: Contribution of a mass element

The usual solution formula for quadratic equations gives the minimum for the solution with the + and the maximum for the solution with the -. The same result is obtained when computing the *eigenvectors* and the *eigenvalues* for the matrix. What we are looking for is a rotation α that makes the 2 by 2 matrix of second moments diagonal. This leads to an eigenvalue problem.

$$\begin{pmatrix} m_{2a} & m_{2b}/2 \\ m_{2b}/2 & m_{2c} \end{pmatrix}$$

If $m_2b \approx 0$ or $m_2a \approx m_2c$ then the zone is too round to determine an orientation. The *roundedness* of the zone can be evaluated as(Horn 1986, 53):

$$r = \frac{a - b}{a + b}$$

$$a = (m_{2a} + m_{2c})\sqrt{m_{2b}^2 - (m_{2a} - m_{2c})^2}$$

$$b = (m_{2a} - m_{2c})^2 - m_{2b}^2$$

Note that these values are derived for zones but also apply to simply connected regions.

7. Set Operations on Zones

Local operations can determine whether two zones intersect or not (Figure 209). A zonal operation determines the intersection area, computing the area in the zone "intersection zone". The intersection of two zones is the logical *and* of the values which qualify for membership in the zone:

z_1 = cond 1

z_2 = cond 2

Intersection z_1 z_2 =
cond 1 AND cond 2

The use of set operations to determine topological relations is restricted to complement, union, intersection (which are lifted *not*, or, and *and*) (Figure 210). The inclusion of *A inside B* is computed as $A \cap B = A$, if A is disjoint from B then $A \cap B = 0$ (Figure 211). It is not possible to determine touching directly with set operations, but one can determine inclusion and disjointness (a more detailed treatment of topological relations follows in chapter 22xx).

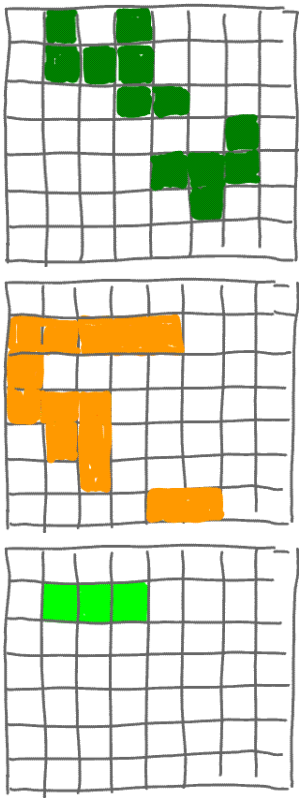


Figure 209: Two zones and their intersection (as Boolean raster)

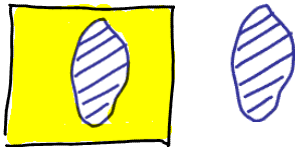


Figure 210: Two figures and their intersection, union and complement of one

Zones are equivalence classes!

8. Summary for Zonal Operations

Zonal operations are selecting areas based on similar values and then compute a value for the whole zone using a second layer. This computed value is then the value for all location of the zone. The operations used for combining the values in the zone are operations that can be used to combine a set of values to a single characteristic value: *sum*, *average*, etc.

Zonal operations are defined independent of the representation and apply equally to raster representation or to

irregular subdivisions. For continuous representation the sum of discrete values becomes the integral, respectively the appropriate finite aggregation operation for this representation(Bird and de Moor 1997).

Figure 211: Two figures disjoint and one inside the other

Many practically useful functions are listed by Tomlin. These functions are only shortcuts for a combination of other functions—and one might argue how the additional cost of learning these functions compares to a building the same functions from few generally useful building blocks. Tomlin gives an example on how several simple and easily understood operations are combined(Tomlin 1990, 163).

We may retain that zonal operations derive a single characteristics of a zone; in the continuous case, this is an integral over the zone (Figure 212), for the discrete case it is the sum over the zone. Zones can be seen as objects and zonal operations as a method to obtain summary properties of an object: area, centroid, axis, etc. The operations given here for a *2d* case can be extended directly to *3d* volumes. They can also apply to *1d* (temporal) data or to the combination of spatial and temporal data.

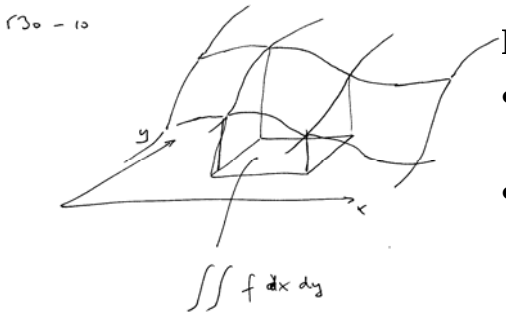


Figure 212: Integral is the sum for an area

REVIEW QUESTIONS

- Review: when is each location a zone by itself? Draw a simple example!
- Proof that the center of gravity of a set of objects is the center of gravity of the parts, each represented as a point mass at its center of gravity.
- Thesis topic: reformulate map algebra with strict mathematics and show what the minimum number of functions is to construct a useful and computationally complete system.

PART FIVE OBJECT DESCRIPTIONS STORED IN A DATABASE

A GIS consists of observations of the properties of objects in the world. People identify objects and observe properties that characterize them and that remain invariant under occurring transformations. A GIS is used to store descriptions of such objects. The first part of this book has concentrated on observations of properties of points in space and time. The second part of the book has a focus on objects, their position in space and time and other properties. New in this part is that objects are not isolated, but related to other objects.

Objects are related to other objects.

The GIS stores facts describing objects. Facts can be the measurements from a surveying operation, where distances and angles between points are observed, it can be the recording of temperature at a location, but it can be the results of derivation from primary facts, like the number of people living in a town, a social index describing the population, or the cadastre, recording ownership relations between people and land.

First half of book: a continuous world: measurements and derived quantities describing points in space-time. Second half of book: a world of objects: Representation, manipulation and storage of objects, their properties and the relations between them.

This part five starts with the database concept: methods to represent measurements and relations between objects and permanently store them. In this part, four issues are considered:

- Generalizing and centralizing storage: the data in a GIS are stored only once and are available to many different applications (database concept).
- Representation of objects and the relation between them and the quantities describing them, together with functions to access the data (data model).
- Permanence of storage: the data is stored such that it is preserved after the close of a program and is available to be processed by the same or other programs concurrently or later (transaction concept).
- Logical consistency of the facts stored in the database.

Chapter 15 CENTRALIZING STORAGE: THE DATABASE CONCEPT

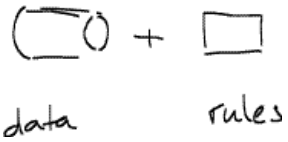


Figure 213: Computational models are data and rules

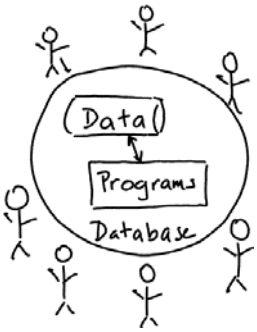


Figure 214: Data storage and programs managing the data serve many users and are a valuable resource

A large number of interdependencies make programming difficult and will eventually bring maintenance to a standstill generalizes:
Complexity is the enemy!

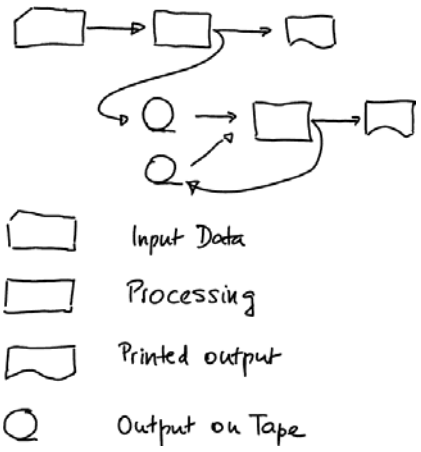


Figure 215: The Input-Processing-Output paradigm for file based data processing in the 1960s.

Information systems are computational models of the world (see chapter 3). They consist of data and rules connecting the data (Figure 213). Databases serve as central repository of data, which controls the *resource* data (Figure 214). The development of databases was initiated by commercial applications and the terminology is influenced by administrative data processing. Administration stores '*records'* of relevant administrative decisions and facts; the records in a GIS are descriptions of observations of the real world and I prefer the notion *fact*.

1. INPUT-PROCESSING-OUTPUT IN THE EARLY YEARS OF ELECTRONIC DATA PROCESSING

Databases were invented in the 1960s(ANSI X3/SPARC 1975). Data processing then was following an *Input-Processing-Output* paradigm (IPO) in which programs depend on each other (Figure 215). Computations required so many steps and connections between the steps where so numerous that any change in one dataset propagated through all the others. The flow graphs for the data processing in a Swiss Bank in 1968 covered a wall! Changes were costly and eventually impossible. The bank initiated a project to build a communication network and a central repository, but this was too ambitious a project for the 1970s and failed. The principles they followed were valid and are the concepts of today's data processing: central repository for data and networked access; unfortunately, the effort was premature and the technology not ready!

2. DATABASE CONCEPT

Organizations, like government agencies, corporations and companies, but also towns rely on large collections of data for their operations. The database *centralizes storage* and *controls access* to all the data in an organization (Figure 216). The database concept assures that all data is potentially available to all parts of the organization and that all programs used the same routines for access and writing to the data. This makes the

valuable resource that the data are available for the whole organization!

To use a database for programming a large, complex data processing system is a substantial conceptual change from the previous Input-Processing-Output model(Bachman 1973; Codd 1982). The linear flow of data through processing units, where data was transformed a record at a time (Figure 215), is replace with a central repository for all data and all programs access the data from this central repository (Figure 216). This has consequences for the structure of data processing in an organization. It changes the way application programs are written.

Data that are centralized and independent from the programs are a resource for an organization.

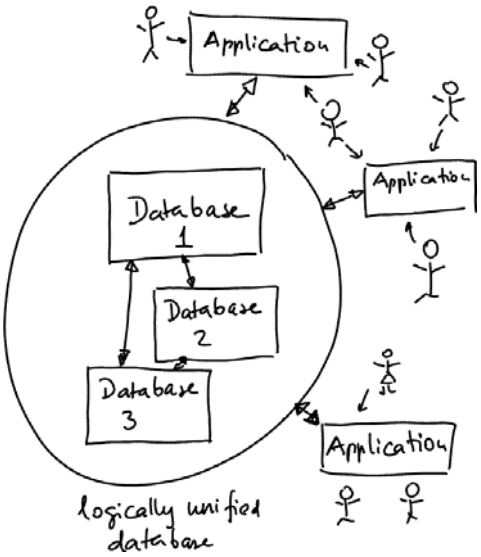


Figure 216: Centralized data as a resource

2.1 CENTRALIZATION

The *centralization* of data in a single unit makes programs *independent* from each other and only dependent on the database (Figure 216). A database is a single logical unit. The data stored is available to all programs through the same interface. It is not necessarily stored in a single unit—storage can be decentralized and even duplicated, but for the programmer this distribution is transparent and managed automatically by the database (Figure 216).

2.2 UNIFORM MANAGEMENT OF DATA

A database is not just a collection of records! Compare with a bank or a library: these are not just collections of money or books, like the money jar in the kitchen or the book shelf in the living room (Figure 217). Banks and libraries have guards that follow rules that control the flow of money or flow of books in and out of the bank or library (Figure 218). Without control of the flow, a bank or library would deteriorate and could not fulfill its function. Similarly for databases: substantive efforts are necessary to guarantee that the data will be always available. The database management system controls the central repository of data.

Figure 217: Money box

2.3 REDUCTION OF DUPLICATE STORAGE

It was observed in the early days of GIS, that the same date elements were stored multiple times in different files and wasted storage space, which was, at that time, expensive. Today, saving in storage is not the primary justification for the organization of a database. Databases are necessary to achieve sharing of data. It

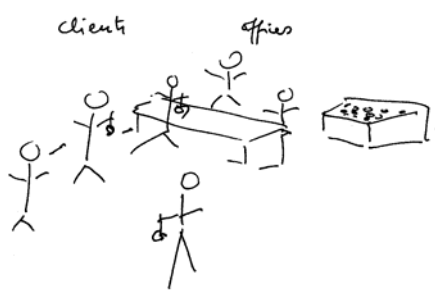


Figure 218: Bank

was also found that much of what on first sight looks like duplicate storage are just data collections that use very similar definitions, but slightly different definition and have therefore slightly different contents. Subtle differences in definitions in laws and regulations are usually the cause.

2.4 DATA SHARING AS MAJOR REASON FOR CENTRALIZED DATABASE

When multiple users need the same data, why do we not simply provide everyone with a copy of the data? This works well if the data are not changed or changes slowly: a digital terrain model, for example, can be distributed as a copy. If data changes, copies do not automatically reflect the changed situation and lead to errors, if the users depend in their decision on the current state. For example, only one copy of cadastral records should exist and all changes inserted there. If multiple copies are updated independently, a fraudulent owner can sell his property twice, once by recording the sale in one registry, and a second time, recording it in another registry! Sharing of data gives instant access to the changes somebody else has applied(Bachman 1973).

Having the data stored once and accessible for all potential users (Figure 216) assures that the data used is up to date: there is only a single copy and anybody using or updating this data must access the same copy. Confusion in the organization resulting from copies representing the same facts in different ways is impossible.

The sharing of updated, "life", data is the major reason for logical integration of data in a database management system.

2.5 ISOLATION

The database management system *isolates* the management of the data from the processing of data in the programs (Figure 219). The database concept integrates the management of the data in a single unit and separates it from application programs. These programs access data through a standardized interface—no program can directly change the physical storage, but all must pass through the database manager (Figure 216). All programmers see the same *logical view* independent of the organization of the storage and their view remains the same even when physical storage changes.

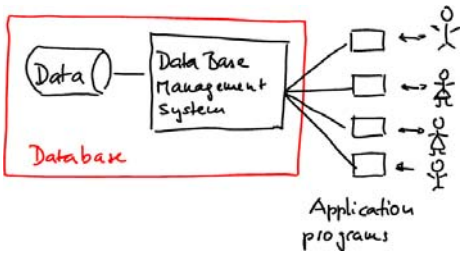


Figure 219: Database consists of Database Management (DBMS) and Data

2.6 A GENERAL DATABASE MANAGEMENT SYSTEM (DBMS)

To construct integrated and consolidated repositories for data of an enterprise is a complex task. It is encountered in similar form and it is effective to produce a generalized program. The database management system (DBMS) is a commercially available software that is adapted (Figure 219) to the task of managing the data collection of an enterprise. The development of generalized DBMS has led to a systematic accumulation of knowledge and solutions are documented in an extensive literature in Journals, like ACM Transaction on Data Systems (TODS), and Conferences like the Very Large Database Conference (VLDB) or the Principles of Data Symposium (PODS). Specialized for GIS is the SSD and now SSTD conference [ref missing xx].

DBMS = Database Management System

2.7 DATA DESCRIPTION LANGUAGE AND DATA MANIPULATION LANGUAGE

The database is constructed from a general set of routines that are specialized in a compilation like process to work with the data of an organization (Figure 220). A description of the data—the logical and physical schemata— are written in the Data Description Language (DDL) using a data model (see next chapter). These descriptions list the object types and the properties the database should store. The compiler translates these descriptions in programs that are then used to store and retrieve the data.

DDL = Data Description Language

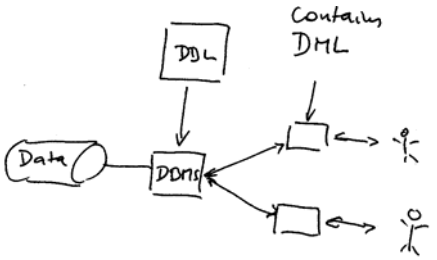


Figure 220: Data Description Language and Data Manipulation Language

The application programmer accesses the data in his program text with statements of the Data Manipulation Language (DML) based on the same data model as the one used for the data description. An augmented compiler for the programming language then compiles the program text with these statements and generates the code that accesses or changes the stored data.

2.8 THREE SCHEMAS (VIEWS)

DML = Data Manipulation Language

The data descriptions are separated in three *schemas*, each describing different aspects of a data collection. These views were standardized early (ANSI X3/SPARC 1975) :

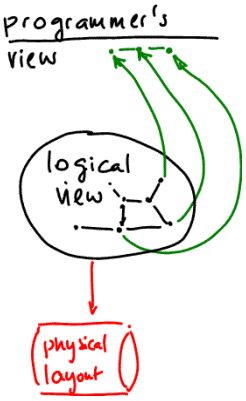


Figure 221: 3 schemas

- Logical schema: A comprehensive, but abstract description of all data in the database. It lists all the data for the whole enterprise and the consistency constraints for them.
- Application schema: It represents the programmer's view, which requires only those data elements that are visible and used by an application. It is a sub-set of the logical schema. It can hide data from a program to enforce privacy rules.
- Physical schema: describes how the data is physically stored.

This separation relieves the application programmer from the need to know the physical storage structure or about data he is not using. Only the relevant part of the logical structure of the data, as presented at the application programmer interface, is included in the programmers view. Compare this to a library: a user has only to give the 'call number' of the book he desires, it is not necessary to understand the organization of the stairwells, elevators, rooms, and shelves where the books are located.

2.9 Performance of Databases

The storage and retrieval of data seems a relatively simple task, but to manage data collections for many concurrent users is difficult:

- Databases are so large and valuable, that they are stored on permanent storage—hard disks. Access to data on a hard disk is slow compared to the access to data stored in main memory.
- Many parallel users must access and possibly change the same data, but maintain consistency (see chapter 17).

Performance is influenced by:

- (spatial) access methods(Samet 1990; Samet 1990),
- buffer management(Reuter 1981),
- query execution strategies, and
- transaction management(Gray and Reuter 1993).

These performance topics are not further discussed here but they influence the design of commercially available DBMS.

*Access to data stored on hard disk takes 10 **milliseconds**. Access to data stored in main memory 100 **nanoseconds**. Accessing data on disk is 10^8 times slower than in RAM; this is the same ratio as between one second and one year! This ratio seems constant and not affected by changes in the technology; it was about 10^7 two decades ago.*

3. Data Models

A data model describes the tools to describe the world, more precisely, the representation of the subset of the world of concern in the application area (which is a system in the sense of chapter 3). The data model lists the concept available to describe the representation and limits indirectly what aspects of reality can be carried over into the computer representation of reality. This applies to all three levels of the schema, but our focus is on

the logical and application schema. Two questions must be answered by a data model:

- How to construct representations of objects?
- How to model the relations between objects? (see next chapter)

Data models give the tools we use to model reality—they are not models of reality.

In the early '80s it was observed that the same concerns appear in the database community—where they were called data models —but also in the artificial intelligence and programming language research community. A conference documented the different points of view(Brodie, Mylopoulos et al. 1984):

- Administrative (DB) programming: few types, many occurrences; permanence of data.
- Artificial Intelligence: many types, with few occurrences per type. Limited lifespan of data.
- Programming languages: few types, few occurrences, limited lifespan of data.

4. HISTORIC DATA MODELS

Admiral Grace Murray Hopper was one of the pioneers of electronic data processing and promoted the use of computer data processing in the US Navy for administration and logistics in the 1950. She was instrumental in the development of the programming language COBOL designed for administrative data processing, which organized data in logically connected pieces, called records, which consists of hierarchically nested fields.

```
record Person
     field Name
          field FirstName
          field FamilyName
     field Address
          field StreetName
          field BuildingNumber
          field Town
```

The CODASYL network data model extended these structures for data and introduced connections between records, so-called 'CODASYL sets'(CODASYL 1971; CODASYL 1971). It was widely used for administrative applications (Figure 222).

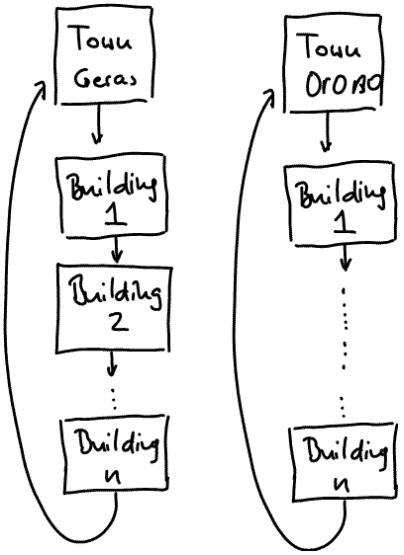


Figure 222: Two CODASIL sets with records for town with their buildings

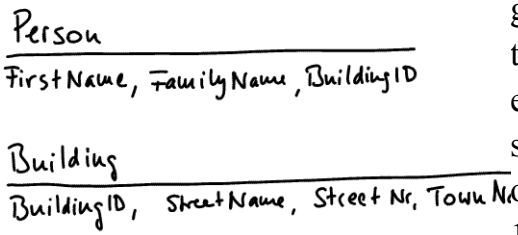


Figure 223: Relational Schemas

Models that are close to the application but not formal make the translation difficult.

The relational data model (Codd 1970; Codd 1982) dominates database applications since the 1990s. The schema describes these tables (Figure 223). Data is arranged in tables and the schema gives the head of the table (Table 3). Researchers introduced object-oriented concepts (Atkinson, Bancilhon et al. 1989; Lindsay, Stonebraker et al. 1989; Stonebraker, Rowe et al. 1990) and proposed object-oriented data models, which overcome some of the limitations of modeling with the relational data model(Codd 1979; Deux 1989; Bancilhon, Delobel et al. 1992; Tansel, Clifford et al. 1993).

The development in GIS followed similar lines of development as the commercial applications. Data storage in GIS started as independent files, with proprietary structures optimized for the application programs used. Later database systems were used for storage of the administrative data, but the geometric data continued in proprietary file structures because the computer systems and databases of the time were not fast enough(Frank 1988). Only in the late 90s standard database systems were extended to include methods for special treatment of spatial data, which were proposed earlier(Frank 1981; Samet 1990). This permits today the integration of all the geographic data of a GIS in a single standard database.

FirstName	FamilyName	StreetName	BuildingNumber	Town
Peter	Artner	Hauptstrasse	13	Geras
Susi	Artner	Hauptstrasse	13	Geras
Karl	Artner	Hauptstrasse	13	Geras
Sabine	Artner	Hauptstrasse	13	Geras
Max	Egenhofer	Grove St	28	Orono
Andrew	Frank	Vorstadt	18	Geras
Stella	Frank	Vorstadt	18	Geras
Astrid	Frank	Vorstadt	18	Geras

Table 3: Example Data

5. CONCLUSION

A database, builds a model of reality representing the "knowledge" an agent has about the world. The data description for a database is expressed in a data model, which is a sort of data language. If the data model is closer to the conceptual or cognitive models, it is easier for the designer to produce an appropriate database schema(Booch, Rumbaugh et al. 1997). The translation of her view of reality to a formal description is more direct and requires fewer steps. It is likely that the model contains fewer errors. If the modeling language is closer to a computer implementation, the database is more likely achieving acceptable performance. In the past, modeling tools were more

influenced by implementation consideration, the object model in C++ (Stroustrup 1986) is perhaps the most recent and extreme example.

Models, which are close to implementation make the task of the analyst difficult and contribute to the 'software crisis'.

Only solutions that have a convincing, simple algebraic structure endure: good theory remains for decades or centuries. The relational DB theory, which has a mathematical foundation, remained for more than 20 years and we will present in the next chapter a mathematically inspired simplification of it. Ad hoc solutions are rapidly superseded by ‘new and improved versions’ produced by companies or standardization committees.

Three aspects to retain:

- a language to describe data and how it can be accessed, independent of programming languages (see chapter 16);
- consistency of the data can be controlled by the database through the transaction concept; (see next chapter 17);
- a logical data description is separated from the description of the physical storage (ANSI SPARC).

REVIEW QUESTIONS

- What is a data model?
- What are the 3 levels in the ANSI/SPARC/X3 model?
- DML and DDL—what are they?
- Explain the difference between logical and physical centralization.
- What is meant by the expression 'sharing life data'? Why is it important?
- Why centralization of data storage? What is achieved?
- What is the reason that DBMS are technology dependent? What is the performance issue in a database management system?

Chapter 16 A DATA MODEL BASED ON RELATIONS

Data stored in a central repository must be accessible in a *uniform* way for all programs. Data structures built for special programs can be optimized for particular uses, but this cannot work, if data is centralized and used by many programs. A uniform method of access must satisfy the different requirements of all programs alike. To achieve flexibility a mathematically clean data model is necessary.

> "A data model is a combination of at least three components:
> (1) A collection of data structure types …
> (2) A collection of operators or rules of inference…
> (3) A collection of general integrity rules" (Codd 1982, 395/396)

Figure 224: A overlap B

The relation data model described here is based on the concept of function, generalized to '*relation'*. It is a further development of the classical Data Models, like the relational (Codd 1970; Codd 1979; Codd 1982) or Entity-Relationship (ER) data model(Chen 1976).

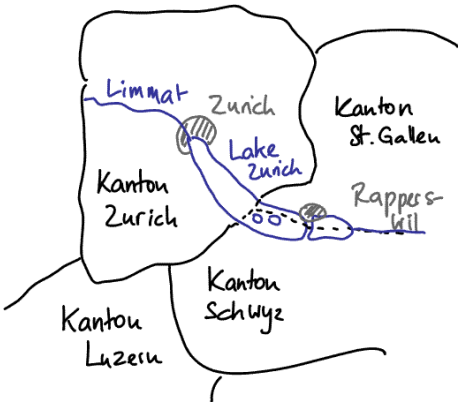


Figure 225: Kanton Zürich overlaps Lake Zürich

1. RELATIONS

Figure 224 shows two regions A and B which overlap; this is a relation between them. The geographic relations in Figure 225 are numerous: the lake of Zürich overlaps with the Kanton Zürich, Kanton Schwyz and Kanton St. Gallen; we see that Kanton Zürich and Kanton St. Gallen are neighbors, another relation, etc.

overlap (lake Zürich, Kt. Zürich)
overlap (lake Zürich, Kt. Schwyz)
overlap (lake Zürich, Kt. St. Gallen)

neighbor (Kt. Zürich, Kt. St.Gallen)
neighbor (Kt. Zürich, Kt. Schwyz
neighbor (Kt. Schwyz, Kt. St.Galle

We will write relations as predicates, that is, functions with two arguments yielding a Boolean result (many texts use *a R b* for the predicate *R (a,b)*). Relations can have several arguments, but relations with more than two arguments can be split into binary relations. For example the relation

parents (Andrew, Irja, Stella)

is split in two relations

father (Andrew, Stella)
mother (Irja, Stella).

It is thus sufficient to develop the theory for binary relations only. We will here always understand 'binary relation' when we use the term relation. Relations have a converse: *r: A -> B* has the converse *r': B -> A*. This is the major difference to functions, which have not always an inverse. The relations which are functions are called *simple*: if the relation *rel: A -> B* is a function, then

rel (a,b) = True <=> rel (a) = b.

Die Welt ist, was der Fall ist. (Wittgenstein 1960)

2. FACTS AND RELATIONS

The representation in a data model conceptualizes the world as entities and facts that describe the entities and the relations between the entities.

An entity is anything conceptualized as having an independent, permanent existence

- **Entities** are the conceptual units, the "objects" in a broad sense. We collect information about these entities. They are represented in the database by identifiers (abbreviated as ID) , which are unique like entities are unique. There are no copies of me!

Entities are represented by identifiers.

- **Facts** assign some properties to an entity, for example a measurement value as the result of an observation. My weight is 80 kg. Facts are a (generalization) of a measurement; prototypically they are the result of an observation of an aspect of an entity, but they can also describe some derived properties of the entity. A fact links a value to an entity in a relation. Facts describe not only properties of entities, but also relations between entities.

Facts describe entities.

Relations are collections of facts of the same type.

A database is a collection of relations.

This is a most general approach to recording the knowledge we have about the world. In this *relation* (not relational) data model the database is a collection of relations, which consists of facts (Figure 226). Identifiers (ID) represent the entities. The database manages the ID and assures that they are unique in the context of the database. This data model is a refinement of the ordinary relational model(Codd 1970): it is restricted to binary relations and system controlled IDs are used to identify the entities.

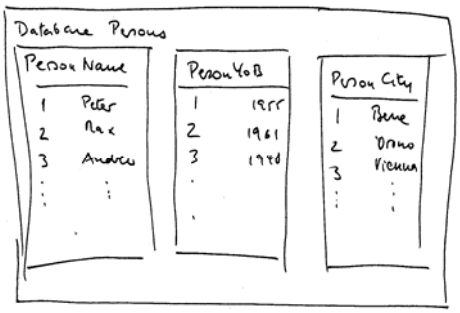


Figure 226: Relation Database

Because we model facts describing entities, all relations have the form *ID -> value* (remember: ID is a special case of value, *ID -> value* includes relations with the signature *ID -> ID*). A fact makes a statement about an entity and a relation it has to some value or to another entity (which is a relation *ID -> ID*). Many of the relations used to describe real world entities in a GIS will be functions, but not all; relations give the generality to cover all cases uniformly. For example, one might think, that ZIP -> townName is a function (or perhaps townName -> ZIP), but neither is the case in any country I know of, and we can only give a relation ZIP -> townName) (Table 4 for an Austrian example).

1010	*Wien*
1040	*Wien*
1050	*Wien*
2093	*Geras*
2093	*Fugnitz*
2093	*Pfaffenreith*

Table 4: Part of the relation between ZIP and name of town in Austria

3. Observations as Relations

The data in a GIS represent measurements. These are observations of some property of the real world. For example, we observed this morning at 07:12 the exterior temperature at the airport Vienna-Schwechat and the result was *15.4 °C*.

The entity is the space-time point at which a set of observations were made. This space-time point has the properties:

- the type of observation: temperature
- the location: Airport Vienna-Schwechat (outside)
- the time: July 9, 2004, 07:12
- the value: *15.4 °C*

These measurements are all facts that describe properties of entity *temperature this morning*. These relations are all functions, because only one value belongs to a single space-time point for each of these observations. Assume that the space-time point entity at which the observations were made, has the ID 23411 then we can construct four functions:

```
observation_Type (23411) = exterior temperature
location (23411) = Airport Vienna-Schwechat
time (23411) = July 9, 2004, 07:12
value (23411) = 15.4 °C
```

4. Example Relations

The example data introduced in Table 3 of chapter 15 is broken into tables, each representing a single function. We have to introduce entities and the corresponding entity identifiers. We select *P1, P2, P3,* and *P4* for the person's identifiers, *H1, H3*,

and *H4* for identifiers for homes, *S1, S3, S4* for the street identifiers, and finally *T1* and *T2* for the town identifiers.

Person -> FirstName	
P1	*Peter*
P2	*Susi*
P3	*Max*
P4	*Andrew*
P5	

Person -> Home	
P1	*H1*
P2	*H1*
P3	*H3*
P4	*H4*

Home -> Street	
H1	*S1*
H3	*S3*
H4	*S4*

Street -> Street-Nr	
S1	*Hauptstrasse*
S3	*Grove Street*
S4	*Vorstadt*

Home -> StreetNumber	
H1	*13*
H3	*28*
H4	*18*

Street -> Town	
H1	*T1*
H3	*T2*
H4	*T1*

Town -> Name	
T1	*Geras*
T2	*Orono*

Town -> ZIP	
T1	*2093*
T2	*04469*

Table 5: The Example Data as relations

5. RELATION ALGEBRA

Knowledge about the world is stated as the existence of relations between entities and logical rules are used to combine such relations. Predicate calculus can be used (chapter 4). The tables above each represent such a relation: for example

```
personName (P1, Peter).
```

An algebraic treatment was suggested by Schröder in the late 19th century (Schröder 1890) and Tarski gave relational calculus is present form (Tarski 1941). This is the mathematical foundations for the relation data model presented here cast into a categorical framework(Bird and de Moor 1997).

5.1 INTENSIONAL AND EXTENSIONAL DEFINITION OF A RELATION

Like functions (chapter 11), relations can be defined *intensional* with a general rule, a formula. For example: the relation square is all value pairs (x, x^2), but also *equal*, *lessThan,* etc. are relations. More common and the situation for which databases are requires is to store facts by enumeration in tables. This is said to define relations by *extension* (extensional definition, Table 5).

Relations in a database are usually defined extensionally by tables.

5.2 THE CONVERSE OF A RELATION

If a relation *R* relates *a* to *b* then the converse relation *C* relates *b* to *a*. The domain of the converse of a relation is the codomain of the relation; the codomain of the converse relation is the domain of the relation.

a R b < = > b C a	*C = conv R*
conv . conv = id	
dom R = codom C	*dom . conv = codom*
codom C = dom R	*codom.conv = dom*

Relations are a generalization of functions; they have a converse. This makes relations different from functions, which have not always an inverse; only functions that are injections have one (see chapter 5xx). Relations and its converses are dual categories to each other.

```
Rel :: a -> b -> Bool
ConvererseRel :: (a-> b-> Bool) -> (b -> a -> Bool)
```

For example, the relation *inside* between an island and the lake is a function, because an island is in exactly one lake. The converse, the relation *contains* between the lake and the islands are not a function: the lake of Zürich contains two islands, Ufenau and Lützelau, and functions must always have a single element as the result (Figure 225).

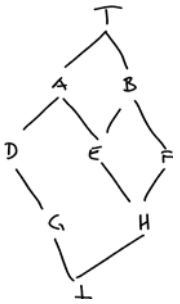


Figure 228: Partial order; elements that are comparable are connected, the larger one above the other

Contains: Lake -> Island	
Lake Zürich	*Ufenau*
Lake Zürich	*Lützelau*

converse: isContainedIn : Island -> Lake	
Ufenau	*Lake Zürich*
Lützelau	*Lake Zürich*

Person -> FirstName	
P1	*Peter*
P2	*Susi*
P3	*Max*
P4	*Andrew*

The converse: FirstName -> FirstName	
Peter	*P1*
Susi	*P2*
Max	*P3*
Andrew	*P4*

Table 6: Relations and their converse

5.3 ORDER BETWEEN RELATIONS

The definition of a relation as a table, which is a set of pairs, gives an order relation by inclusion. A relation *r* is included in another one *p* ($r \subseteq p$) if all the values of *p* are also values of *r*. For example, in a table representing a relation one just deletes a few rows and gets a smaller relation, which is included in the first one (Table 7).

This order relation is only a partial order (see next section), because not every relation can be compared with any other. Order relations induced by $\subseteq$ have a dual, namely the order

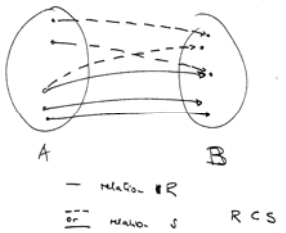


Figure 227: Two relations $S \supseteq R$ (S includes R)

induced by $\supseteq$. To be precise this can be called the "order dual" and must be differentiated from the categorical dual of a relation and the converse (Mac Lane and Birkhoff 1991, 145). Inclusion is distributive with respect to composition:

(s1$\supseteq$ s2) and (t1 $\supseteq$ t2) => (s1.t1) $\supseteq$ (s2.t2).

p: Person -> FirstName	
P1	*Peter*
P2	*Susi*
P3	*Max*
P4	*Andrew*

r: Person -> FirstName	
P1	*Peter*
P3	*Max*
P4	*Andrew*

Table 7: A relation p $\supseteq$ r

For relations we have a minimal element, namely the empty relation, and a maximal element, namely the relation that is true for all entries. Consider the relation neighbor (Figure 225) between the Cantons Zürich, Luzern, Schwyz, and St. Gallen. The cardinality of this maximal relation is the product of the cardinality of the domains; in this case the relation *neighborAll* :: *Kanton -> Kanton* has cardinality *4 * 4 = 16*. The relation neighbor is much smaller and has only 10 entries: *neighborAll* $\supseteq$ *neighbor* $\supseteq$ *empty*.

neighborAll

- *Zürich*
- *Zürich*
- *Zürich*
- *Zürich*
- *Schwyz*
- *Schwyz*
- *Schwyz*
- *Schwyz*
- *Luzern*
- *Luzern*
- *Luzern*
- *Luzern*
- *St. Gallen*
- *St. Gallen*
- *St. Gallen*
- *St. Gallen*

- *Zürich*
- *Schwyz*
- *St Gallen*
- *Luzern*
- *Schwyz*
- *Zürich*
- *St.Gallen*
- *Luzern*
- *Zürich*
- *Schwyz*
- *St.Gallen*
- *Luzern*
- *Zürich*
- *Schwyz*
- *Luzern*
- *St.Gallen*

neighbor

- *Zürich*
- *Zürich*
- *Zürich*
- *Schwyz*
- *Schwyz*
- *Schwyz*
- *Luzern*
- *Luzern*
- *St.Gallen*
- *St.Gallen*

- *Schwyz*
- *St allen*
- *Luzern*
- *Zürich*
- *St.Gallen*
- *Luzern*
- *Zürich*
- *Schwyz*
- *Zürich*
- *Schwyz*

6. PARTIAL ORDER AND LATTICE

Assume a set of objects L with a partial order $\supseteq$. The maximal element is the top ($\top$) and the minimal element is the bottom ($\perp$), if they exist. Partial orders and lattice are highly regular theories that expose *duality*, which will be used here and later.

Poset (partially ordered set) ordered by $\leq$	
Reflexivity	$l \leq l$
Antisymmetry	$l \leq m \;\&\; m \leq l \Rightarrow l == m$
Transitivity	$l \leq m \;\&\; m \leq n \Rightarrow l \leq n$
Units (if exist)	$l \leq \top,\ \perp \leq l$

6.1 UPPER AND LOWER BOUND

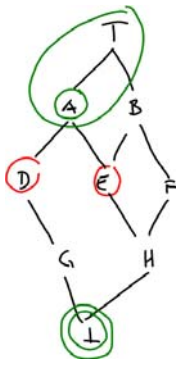


Figure 229: Upper and lower bound for D and E (with lub D E = d join E = A and glb D E = D meet E = ⊥)

The elements above an element *x* (e.g., in Figure 229 for *D* it is *D* and ⊤) and the element lower than *x* (e.g., in Figure 229 for *D* it is *G* and ⊥) are sets. Elements that are not connected by an arrow are not directly comparable; they may be indirectly comparable, using transitivity of the order relation. These sets contain all the elements that are directly comparable with a given element and are larger (smaller).

"In many posets (partially ordered sets) one can define two operations somewhat resembling multiplication and addition" (Mac Lane and Birkhoff 1991, 144). The *upper bound* of two elements is the set of elements that are above both of them and correspondingly for the *lower bound* (by duality):

upper bound (x,y) = {m | m ≤x, m ≤y}
lower bound (x,y) = {m | m≥x, m ≥y}.

The upper bound may have a *unique* least element (*least upper bound, lub*) and the upper bound a unique greatest element (*greatest lower bound, glb*) (Figure 229). The operation that gives the greatest lower bound for two elements is called *meet* ∧ and the dual operation that gives the least upper bound for two elements is called *join* ∨.

join ∨
meet ∧
lattice = partial order with unique lub and glb for every pair of elements

a ∨b = least upper bound (a,b)
a ∧ b = greatest lower bound (a,b)

6.2 MEET AND JOIN

A partial order in which for every two elements a least upper bound and a greatest lower bound exist, is a lattice (German: Verband). A lattice <*L*, ∧, ∨> is an algebraic structure on a set *L* of partially ordered elements (by <), where for each pair of elements *l1* and *l2* in *L* exist unique elements *meet* (*l1, l2*) and *join* (*l1 l2*) (Gill 1976, 144; Mac Lane and Birkhoff 1991).

Lattice <L, ∧, ∨>		
idempotent	*r ∧ r = r*	*r∨r = r*
commutative	*r ∧ s = s ∧ r*	*r ∨s = s∨r*
associative	*r ∧ (s ∧ t) = (r ∧ s) ∧ t = r ∧ s ∧ t*	
	r ∧ (s ∧ t) = (r ∧ s) ∧ t = r ∧ s ∧ t	
absorption	*r ∧ (r ∨s) = r ∨(r ∧ s) = r*	
consistency	*r ∧ s >= r*	*r ∨ s <= r*
isotone	*r <= s then r ∧ t <= s ∧ t*	*r ∨ t<=s ∨ t*

Lattices with additional rules are often used; a lattice is called distributive, if the following distributive law holds.

distributive	*r ∨ (s ∧ t) = (r∨ s) ∧ (r ∨ t)*	*(r ∧ s) ∨ t = (r ∨ t) ∧(s ∨ t)*

A lattice may contain elements 1 and 0, such that:

top, bottom	$l <= 1,\ l >= 0$	
identities	$l \wedge 0 = 0$	$l \wedge 1 = l$
	$l \vee 0 = l$	$l \vee 0 = l$

The rules for meet and $\vee$ are dual; one could think of a lattice as a combination of two semi-lattices,(L, $\vee$) and (L, meet), each with one operation, which is idempotent, commutative and associative, combined(Mac Lane and Birkhoff 1991475).

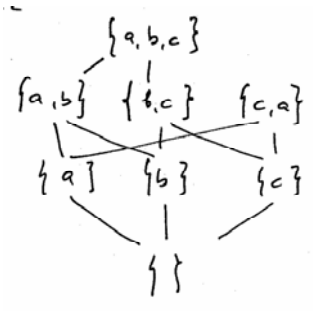


Figure 230: The lattice of the powerset of {a,b,c}, subsets are linked upward to the superset.

6.3 POWERSETS

For a given set of elements, one can consider all possible sets that can be formed from these elements, starting with the empty set, then all the sets with one element (singletons), then the sets with two elements, etc. till one reaches the set with all the elements. The set of all possible subsets is called the *powerset* and expressed for a set of elements U as 2^U.The empty set and the set S are both elements of the power set. The power set is closed with respect to the operations union and intersection (in this context sometimes denoted as *sum* and *product*).

Example: $S = \{1,2,3\}$, $2^S = \{\{\}, \{1\}, \{2\}, \{3\}, \{1,2\}, \{1,3\}, \{2,3\}, \{1,2,3\}\}$

Powerset

$a \cup b \in$ *powerset s,*

$a \cap b \in$ *powerset s.*

$a \cup b = a$ *if and only if* $a \cap b = b$

The powerset with the subset relation is partially ordered and is a lattice. In a powerset, the *join* is the *union* and the *meet* is the *intersection* of sets.

Relations are elements of the powerset of the Cartesian product of the domains. The maximal element, the $\top$ of the lattice, is the relation with all entries, the minimal element, the $\bot$ of the lattice is the empty relation.

7. RELATION CALCULUS

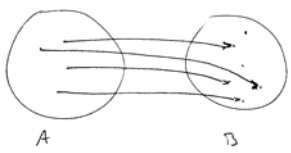


Figure 231: Simple relation

Relations can have properties(Bird and de Moor 1997, 89):

Reflexivity	*a R a*
Irreflexivity	*not (a R a)*
Symmetric	*a R b => b R a*
Assymetric	*a R b => not (b R a)*
Antisymmetric	*a R b and b R a => a == b*
Transitive	*a R b and b R c => a R c.*

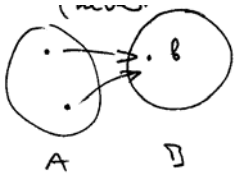


Figure 232: Relation that is not simple

A relation is called **simple**, if for any *b* there is at most one *a*—the converse relation returns one or no element; it is a partial function.

f.R < s = r < f'.S
r.f' < s = R < S. f

A relation is called **entire**, if for any *b* there is at least one *a*—the converse relation returns one or more elements.

If a relation is simple and entire, then it is a total function and has an inverse function.

f . g = f . g' => g = g'.

The relation *child* is irreflexiv and asymmetric. From the examples above, *neighbor* is symmetric, *contains* is transitive.

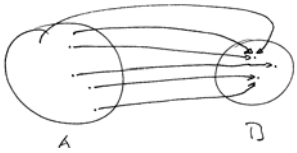


Figure 233: Entire relation

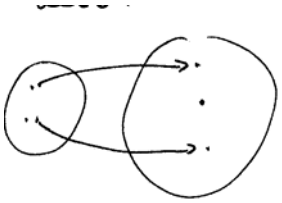


Figure 234: Relation that is not entire

8. ALLEGORIES: CATEGORIES FOR RELATIONS

An allegory is a category with some additional properties. From categories we get composition and identity. Allegories are categories to treat indeterminacy—to an argument there may be several results: Consider the example data in figure xx in chapter xx: who lives in Geras? Possible answers Peter, Susi, or Andrew.

Allegories assume the additional operations:

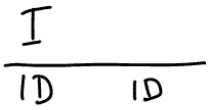
- partial order
- complement
- intersection
- converse
- meet and join

8.1 COMPOSITION

Composition of relations is like function composition; it chains two relations together. It is traditionally written as ";", given that it is equivalent to the function composition in category theory we write "." (Note: *A;B = B.A*)

Traditional: a R b ; b S c < == > a (R;S) c = a T c, where T = R;S
Allegorical: S (b,c) . R (a,b) <==> (S.R) (a,c) = T (a,c), where T = S.R

For example, the relations *p: Person->Home* and *h: Home -> StreetName* can be composed *p.h = ph* to give a relation *ph: Person -> StreetName*. Composition of relations is only defined when the types correspond, i.e., the type of the codomain of the first relation is the type of the domain of the second one (this is like function composition!).



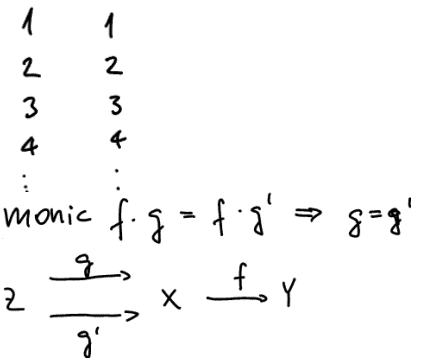


Figure 236Monic Relation

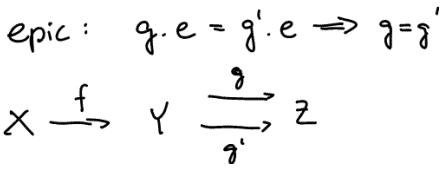


Figure 237: Epic Relation

Injection: every element from the source maps to a different element in the target domain
Surjection: no two elements from the source map to the same element in the target domain.

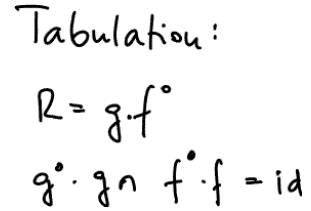


Figure 238: Condition for tabulation

C -> Person	
1	*P1*
2	*P2*
3	*P3*
4	*P4*

C -> FirstName	
1	*Peter*
2	*Susi*
3	*Max*
4	*Andrew*

8.2 IDENTITY

The identity relation, which is true for any a: *I (a,a,)* is the unit for composition.

R = I . R and R = R . I

The identity relation can be imagined as a table, which has for each ID the same ID in the second column (Figure 235).

8.3 MONIC AND EPIC RELATIONS

In the category of relations the concept of injective (see chapter 3) can be generalized. A function or relation *f* is said to be *monic* if for any *g, g′: f . g = f . g′* implies *g = g′*.. For the category of sets, a function is monic if it is an injection.

A relation *e* is *epic*, if for any *g, g′: g. e = g′ . e* implies *g = g′*. For the category of sets, a function is epic if it is a surjection. A bijection is monic and epic

8.4 DUALITY IN CATEGORIES: OPPPOSITE CATEGORY

A category *C* has an opposite category C_{op}, in which all arrows are reversed, that is, for every function *A -> B* in *C*, there is a function f_{op} *B -> A*. The opposite category for relations is dual to the original one. Duality in categories maps monic to epic and epic to monic: if *f* is monic in *C*, then f_{op} is epic; if *e* is epic in *C*, then e_{op} in C_{op} is monic. The dual of an isomorphism is an isomorphism.

8.5 TABULATION

The relations we consider here have tabulations, they are defined extensionally by enumeration as a table; already Codd pointed out, that tables are at a special case of relations(1982).

This can be expressed mathematically: given the relation *R : A -> B* and a pair of functions *f: C -> A* and *g: C -> B* then *R* is tabulation

if R = g . (conv f) and (conv g . g) ∩ (conv f. f) = id

Intuitively, we can see the functions *f* and *g* as functions from the index in the table enumerating the relation r to the first and second column (Table 8).

8.6 PROPERTIES OF RELATIONS EXPRESSED POINTLESS

The properties of relations can now be expressed *pointless* (see subsection 7.1):

Reflexive id < R

Table 8: Two Functions to demonstrate tabulation

Tabulation:

$R = g \cdot f^{\circ}$

$g^{\circ} \cdot g \cap f^{\circ} \cdot f = id$

Figure 239: Condition for tabulation

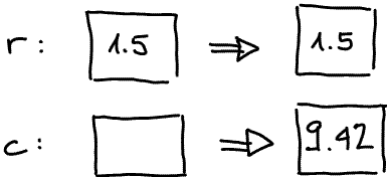


Figure 240: Effect of the execution of the statement X on storage location r and c

Transitive $r \,.\, r < r$
Symmetric $r < conv\ r$
Antisymmetric $r\ meet\ conv\ r < id$
Simple $s \,.\, conv\ s < id$
Entire $id < conv\ r \,.\, r$

9. ACCESS TO DATA IN A PROGRAM

Programs access data values at different times during their execution. This translates to access functions in the program text; at that place, a variable name pointing to the data is inserted in the program text. For example to calculate the circumference of a circle with radius measured as *1.5 cm*, where the circle radius is contained in the variable *r* and a constant π is defined, a formulae like $2\ \pi\ r$ is used and then assigned to a variable *c* in a program statement like (Pascal notation(Wirth 1971; Jensen and Wirth 1975)):

```
c := 2.0 * pi * r
```

The use of the variable name *r* on the right side of the assignment statement is accessing a data value, whereas the variable name *c* on the left side of the statement is an assignment of the computed value to the variable *c*. It changes storage such that the variable *c* can be used to access this new value later (Figure 240). Working with variables in a program assumes—transparent to the programmer—that the variable name serves as two different functions: one to *get* (read) a value and one to *put* (write) a value, depending on which side of the assignment it is used.

If a program is database-oriented, it must get data from the database and write updated values back to the database. Local variables are replaced with functions retrieving the value from the centralized storage and all assignments (in Pascal the ":=" operation) result in storing the new value in the database. One could replace the retrieval with a function *get* and the assignment with a function *put*, which changes the storage. In a language where variables cannot change—like ordinary mathematical notation—the update must create a new value for the database that has all the same content except for the element changed. The above program would then change *memory* to a new value *memory'*:

```
memory'= put (memory, c, (2.0 * pi * get (memory, r))
```

In a program we have a variable name for each value (above *r*); in a database the data is accessed based on identifiers. For example, the birthday of a person is found by first finding the

identifier the database uses for this person given her name and then find the birthday related to this identifier. Codd has coined the expression *'relational complete'* for access methods that permit to find all data using the internal relations between the data(Codd 1982).

10. DATA STORAGE AS A FUNCTION

Data storage can be seen as a function, which produces for an identifier a value:

get (i) = d

or more detailed including the environment, in which the values are stored:

get (memory, i) = d.

If we want to update the value associated with an identifier we produce a new state of the storage (written above as *memory'*) with the function *put*. Memory is seen as the extensional definition of the *get* function; *put* is producing a *new* function. Extending the function with an additional value is adding a tuple *(x, f(x))* to the table stored in memory. The axioms are:

Memory

get (put (memory, i, v), j) = if i == j then v else get (memory, j)
get (new, j) = undetermined.

The hardware used for storing data—hard disk or RAM—have interfaces to read and write data, which can be used to implement such a get and put function. The concern here is the structure of the access function, specifically the arguments necessary.

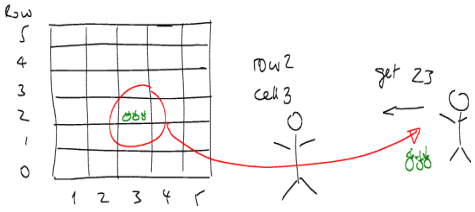


Figure 241: Data storage and retrieval is like a store room! The clerk translates the locker number into a location and retrieves the contents.

Historical comment: the idea to consider database access as a function was first introduced by Shipman (Shipman 1981) and later adapted to the then new programming language Ada(ADA 1983). It did not find much attention, despite its attractive clean structure; the concern for performance and skepticism against functional approaches convinced the database community that it would not lead to a usable implementation.

11. FINDING DATA IN THE DATABASE

11.1 SELECTION AND PROJECTION IN RELATIONS

Operations to select and project values from relations are the building blocks to construct access functions to find data elements stored in the database. A *selection* on a relation takes

some condition and returns the relation that contains only the tuples that fulfill the given condition.

select :: cond -> rel -> rel
select c r = { t | t ∈ r & c (t) = true}

To select all tuples that have as a first value a given value *v* use a condition like *(v==).fst*, which composed of a function to get the first value out of the tuple (respective the second one) and then compare it with the given value.

The *projection* on a relation produces from a set of tuples a set which contains the entire first (respective the second) part.

Examples: consider the relation *pfn: Person -> FirstName*. A selection to find the relation of persons with name Peter is

select ("Peter"==).snd) pfn *result: {(P1, Peter)}*

A projection to the second part in the tuples of the whole relation is: *{Peter, Susi, Max, Andrew}*. Note, that the result of selection and projections are sets; selection produces a set of tuples, which is a relation, projection a set of values.

11.2 ACCESS FUNCTIONS FOR DATABASE

To find data in a database we need functions, which select some elements from a relation, for example, all persons with *name == Susi* (gives *P2*) or all persons living in Geras (gives *{P1, P2, P4}*).

The database maintains that IDs are unique and many relations are functions from ID to a value, we have for relations which are simple (ie., are functions):

toValue :: ID -> relationType -> db -> value.

This function returns the value of the tuple with the given ID; it is a selection applied to the relation as a table and projects from the result the ID. For the general case, when the relation is not a function, the selection returns a set with possibly more than one tuple and the projections give a set of values:

toValue' :: ID -> relationType -> db -> [value].

This is a selection on the *first* part and then a projection on the *second* part of the tuple.

The converse relation is not a function and could be transformed into a function where the result is a set of IDs (this is the power transpose(Bird and de Moor 1997, 103):

fromValue :: value -> relationType -> db -> [ID].

This is a selection on the *second* part and a projection on the *first* part; it is the operation *toValue'* applied to the converse relation (i.e. *fromValue = toValue'. converse*)

Complex queries are composed from elementary queries, but the *toValue* and *fromValue* functions do not compose, *fromValue* produces a set of ID whereas *toValue* or *toValue'* needs a single ID as input. Similar problems poses the composition of *toValue'* and *fromValue*. Composition is only possible if the result type of all functions and the input types are the same. We achieve this by making both functions take sets of values as inputs:

```
to :: db -> [ID] -> [val]
from :: db -> [val] -> [ID].
```

As the IDs stand for the facts, we can say, we may want to find the ID ***from*** some value, or to find the value ***to*** a given ID. This corresponds for *to* follow the direction of the arrow in the diagram (Figure 242) and for *from* to go in the direction opposed to the arrow.

11.3 AUXILIARY FUNCTIONS

These access functions have sets of values as inputs and outputs. Two functions are used to convert sets to single value and back:

- Singleton—converts a single input value into a set
- Unique—converts a set of one element in *Just* this element and *Nothing* otherwise. It is the partial function *converse . singleton,* converted to a total function with the functor *Maybe*. It returns the value *Just v* for queries where the result is – as expected – a single value, and *Nothing* if there is no or multiple values in the result.

12. STORING DATA IN A RELATION DATABASE

In a database with a relation data model, updates in a database consist of inserting or deleting a fact to a relation; a change is a delete followed by an insert, no particular operation is necessary.

These two operations have the signatures:

```
insert:: relation -> ID -> value -> db -> db
delete: relation -> ID -> value -> db -> db
```

DB

toValue' (i, r, new) = []
toValue' (i, r (insert (r, j, v, d)) =
if i==j then v ++ vs else v'
where v' = toValue' (i, r, d)
toValue' (i, r, (delete (i, r, v, d)) =
if i==j then v' \ [v] else v'
where v' = toValue' (i, r, d) (\= set difference)

---

13. EXAMPLE QUERIES

The data given before is described in a diagram, similar to an Entity-Relationship diagram (Figure 242). Arrows indicate relations of a *1: n* type, of which there are two types: one *1:1* leading from an entity to a value, where the tail of the arrow is anchored at the entity, and one *1:n* between entities, where the tail of the arrow is anchored on the side with *1* element. Relations with *n:m* would be shown with a simple line.

Figure 242: The schema for the example data

The person has the attributes name, year of birth, the buildings have a number, street name and number, and the towns have the attributes ZIP and name.

13.1 FIND NAME OF PERSON, GIVEN ID

To find the name of a person to a given ID, means converting the single ID to a list of IDs, then move from this list to the list of Person names (which must be one, or none) and convert it to a maybe value

```
findPersonName id db = unique . to db personName .
singleton $ id
For an input of P3 the result is (Just Max).
```

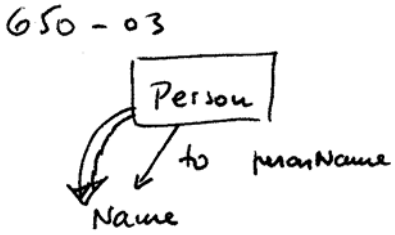


Figure 243: Person to Name

13.2 FIND NAME OF TOWN, GIVEN ZIP

To find the name to a given ZIP code is more involved: we have first to find the town from the given ZIP and then take the result (which is a list of IDs) and get the names to this list. Last check that the result contains only a unique value.

```
findTownNameFromZip zip db = unique. to db townName
                    . from db townZip . singleton $ zip
To an input of 2093 the result is (Just Geras).
```

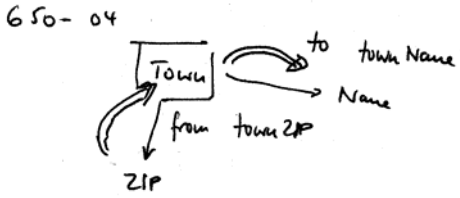


Figure 244: From ZIP to name of town

13.3 FIND ALL PERSONS LIVING IN A TOWN, GIVEN BY NAME

Four steps are necessary to find the names of all persons that live in a town, given by its name:

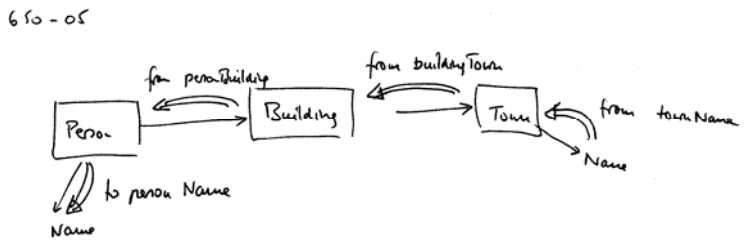


Figure 245: Names of person living in a town given by name

```
findAllPersonNamesIn name db =
                to db personName
               . from db personBuilding
               . from db buildingTown
               . from db townName . singleton $ name
```

We first find the town ID from the given name and then find all buildings in this town (which is also a list of IDs) and then find the ID of the persons living in the buildings, and last, we find the names of the persons involved.

```
For an input of Geras, we find [Peter, Susi, Andrew].
```

14. OTHER DATA MODELS

14.1 RELATIONAL DATAMODEL

The relational data model uses tables as the major structuring element. A relational table collects tuples of arbitrary arity. Each table has a key and the relational table can be seen as a function from key to tuple: *key -> (A × B × .. × M)*. The relational data model is based on relational algebra, for which a substantial theoretical literature exists(Codd 1991).

Operations on relational tables are:

- Select all tuples with a given condition
- Project: retain only some columns of the table
- Join: compose two relational tables using equal values (comparable to the composition of binary relations)

The inputs and result of these operations are tables. This makes relational algebra closed. Closedness was a substantial improvement over previous methods to structure data(CODASYL 1971; CODASYL 1971), which were not explained algebraically and were not closed. Closedness has certainly contributed to the success of the relational data model; we have seen that closedness is also obtained for the relation algebra with *from* and *to*.

FirstName	FamilyName	StreetName	BuildingNumber	Town
Peter	Meier	Hauptstrasse	13	Geras
Susi	Meier	Hauptstrasse	13	Geras
Max	Egenhofer	Grove St	28	Orono
Andrew	Frank	Vorstadt	18	Geras

Table 9: A relational table for person data

14.2 NORMALIZATION RULES

Codd in the original proposal for the relational database model suggested that relations (tables) should be normalized(Codd 1970), to avoid inconsistencies introduced by updates, so-called anomalies in updating(Vetter 1977). Normalization forces a break-up of relational multi-column table if there exist functional dependencies between columns other than the key columns.

Every relation in the relational model represents a Functional Dependency. A functional dependency states that there is a function from the key to the tuple: for each value of the key, there is only one tuple. The key can be a single column or a combination of columns. Functional dependencies describe the intended semantics of the model and are crucial for normalization in relational database theory(Zaniolo, Lockemann et al. 2000). Normalization assures that the functional dependency from key to tuple is the *only* functional or multi-valued dependency in the relational table.

14.3 ASSESSMENT OF RELATIONAL DATABASE

The currently available relational databases are the best solution available for databases. It took more than 10 years between the original publication of Codd's ideas (Codd 1970) and the first viable relational database management systems. Only late in the 1980s the relational database achieved acceptable performance for use in commercial applications.

SQL is intergalactic data speak (Stonebreaker)

The available relational database management systems are highly standardized. Applications written for one DBMS can be transferred with not too much trouble to products of other manufacturers. The clarity of the theory with formally defined semantics has contributed enormously to the popularity of relational database. The query language SQL with Mult-Media Extensions—including spatial and temporal extension—is currently are standardized (ISO/IEC 13249-3:2003).

Occam—make things as simple as possible, but not simpler.

15. ADVANTAGES OF THE RELATION DATA MODEL

15.1 BASED ON ENTITIES WITH ID

The ontological difference between objects and values is fundamental and must be included in the data model. The object "Antares" (the horse I often ride) exists only once and has

permanence in time. We can have many references to it. Values are ideal concepts and many copies exist.

Objects exist only once, representations(values) can be copied.

Codd suggested 'surrogates'(Codd 1979), I use here the term identifier and abbreviate to ID. These IDs are managed by the database and are only used to connect values to entities. The use of database managed identifiers is necessary to manage time varying data: the IDs are the only way to identify an entity through time; other identifiers can change! For example, in many cultures women change their name when marrying.

15.2 CONNECTION BETWEEN ENTITIES

The operations of a relational database allow the connection between any values that has comparable type. Relations are not maintained by the database and the connection between tuples is only based on equality of values in both tuples. This connection is easily lost: The building contains a street address, and the street name relates the building to the street. The connection is broken, if the name of the street is changed and one has not updated all the building records. Similarly, buildings are not found if the name of the street is not spelled correctly. For example, real addresses may contain a “B. Pittermann Platz”, a “Bruno Pittermannplatz”, a “Dr. Bruno Pittermann-Platz”, etc. etc. all referring to the same plaza in Vienna.

Relational tuples contain values only

The relational database schema does not contain the information that these two tables are linked—it only contains the information that the field *streetName* in one and *strName* in the other are of the same type and therewith a join is possible.

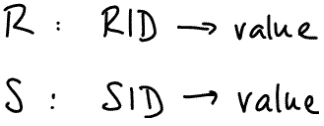


Figure 246: Two relations with the same codomain

The relation based data model connects only through IDs, not values of properties. If a link must be constructed from values, a new relation with a new ID must be introduced (Figure 246, Figure 247)

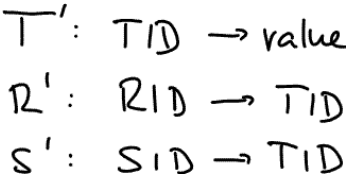


Figure 247: The two relations linked

16. ALTERNATIVES

It is difficult to find agreement on viable solutions to overcome the disadvantages of relational database. Some proposals addressed the syntactic restrictions of First Normal Form: NF2, Not First Normal Form database (Schek 1982; Schek and Scholl 1983; Schek 1985) allow repeating groups and generally groupings within a value in a table.

Currently, databases are offered that are called 'object relational'. They are similar to the relation model using

identifiers to represent objects, but allow multi-valued relations(Stonebreaker 1993).

17. SUMMARY

Relations are an elegant calculus to deal with collection of values not objects; this creates conceptual difficulties for temporal databases, where IDs (surrogates(Codd 1979)) are necessary to maintain the continuation of an object in time(Tansel, Clifford et al. 1993). The commercially used Relational Databases are based on set theory and values. The extensions and improvements added since the initial design have made them powerful, but added also to conceptual complexity.

The major advantage of the relation data model is the full generality. Commercial data models, primarily the relational data model, but also the entity-relationship model and the older network data model imply rules, which are reasonable in most cases but not always. These rules and the special situations in which they do not apply are difficult to identify and then to avoid (see chapter 18). The relation data model leads also to an intuitive query language and a simple data manipulation language.

Last, but not least: The relation data model is a special case of the relational data model. It can be used with any relational DBMS—it uses only binary relations. Composition becomes *join*, projection is one of the operations domain or codomain.

REVIEW QUESTIONS

- What is an entity? What is a tuple?
- What does the relational data model consist of? What the relation model?
- What is the data model for relations?
- What is a fact?
- Explain the difference between categories and allegories?

R, *S*, *T* are relations:

- What is meant with *R*;*S*—give an example.
- What is a lattice? In what sense do relations form a lattice?
- Explain why we can say that relations are ordered? Give an example.
- When is a relation symmetric, when transitive?

- What does it mean to state, that a relational database is value based? Give an example where this becomes visible.
- What is the meaning of a statement like ‘connections between tuples are value based’? Give example.
- Given two relations from Clients to ZIP and Stores to ZIP. Transform them to proper relation form and link Clients to Stores.

Chapter 17 TRANSACTIONS: THE INTERACTIVE PROGRAMMING PARADIGM -

Storing the data in a central repository makes them available for many programs; the data remains after the end of the program that collected them and is available and valid in the future. An example application for the use of a database with GIS is the land registry, where the long-term permanence of entries is of vital interest to all land owners!

Making data permanent is the essential architectural step towards interactive computing as we know it today(Bachman 1973). In the age of batch processing, data was printed and the lists were distributed to who ever needed the information. This is not acceptable today: we want to use the computer to search for the data we need now and present the latest information at our screen. The database concept made interactive computing for many applications possible.

1. INTRODUCTION

Programming under the Input-Processing-Output paradigm starts with a sequence of input records that are transformed or merged and results in an output sequence of records (fig. 270.01xx earlier). This processing was batch oriented, that is, all inputs are collected and treated at once, for example once per day or month and the result are distributed to the users in form of a listing.

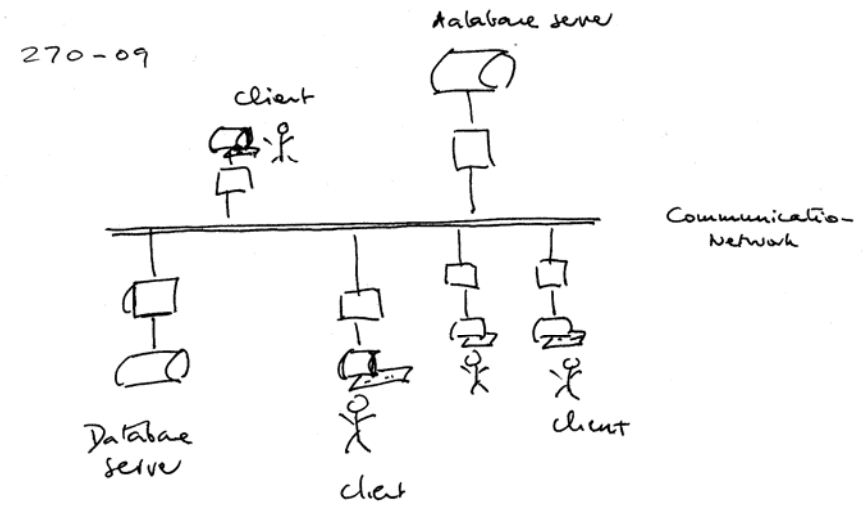


Figure 248: Database in a network with many users

Modern computing is interactive, where the user starts arbitrary operations—some of which result in updates other are simply requests for information—and expects immediate responses on her terminal or connected personal computer. This paradigm of interactive computing is only possible with a central repository of data that is available for all users, and that can perform updates and queries for many users concurrently. This central repository of data is connected through a network to the workstations of the users (Figure 248).

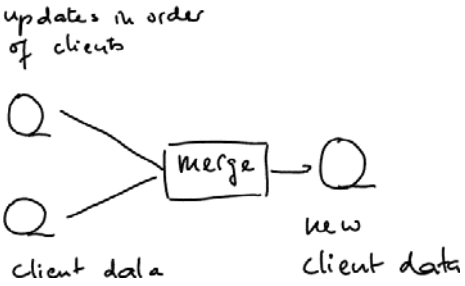


Figure 249: IPO processing: Merging two sequential files

2. PROGRAMMING WITH DATABASE

Programming with the Input-Processing-Output (fig 270-01 in chapter 15) paradigm is dominated by the structure and sequence of elements in the input files. It is a transformation of the sequence of records in the input, one by one, or it is a merge of two sequences of records (Figure 249). The output file is produced in nearly the same order than the input. Occasionally input files are sorted to achieve a faster processing. If something goes wrong, the process is stopped, the error corrected and the process started afresh. This translates to programs that read input files sequentially, record by record, and write output files in a similar manner.

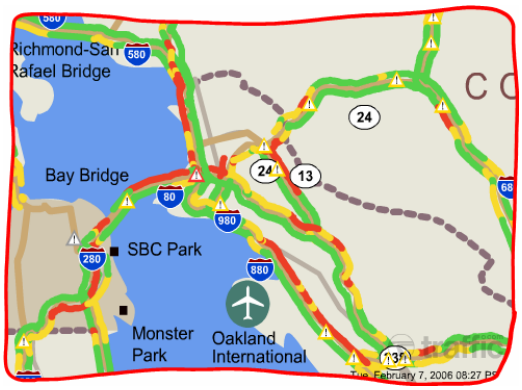


Figure 250: the traffic situation in the Bay Area

Users expect today immediate answers from their computers. Changes in the world are observed and recorded in the database as they occur. For example, we can access maps showing the actual traffic conditions on the highways in some metro areas of the USA (Figure 250).

This interactive paradigm means that a client process interacts with the database process. Changes are processed one at a time. The client processes access data randomly in a pattern, which is not predictable. The result of an interaction is an updated state of the database. Many users interact with the database at the same time.

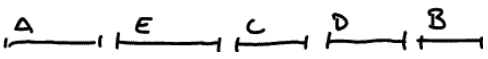


Figure 251: Sequential Processes

3. CONCURRENCY

Database must be prepared to deal with concurrent users. Sequential processing would restrict the access to the database to a single user at a time (Figure 251). It is not acceptable that other users must wait till the first user has finished and thus concurrency (Figure 252) is required for a GIS data server.

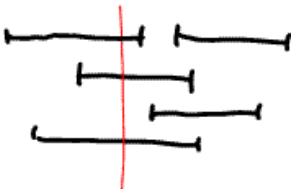


Figure 252:Concurrent Processes

A single computer cannot really execute several programs at the same time, but the results of several processes executed in a single time-sharing system appear as if they were progressing in parallel. Real parallelism of operations is only possible if several processors cooperate as shown in Figure 248. Single processors simulate parallel processing using time-sharing: they execute some operations for a first process, then stop this process and advance another process, then go to another one, etc. till returning to the first one and advancing this one.

Definition of concurrent: More than one process is started before all other are ended.

4. THE TRANSACTION CONCEPT

The consistency of a database is threatened during updates—just accessing values for reading does not change the database and will not change a consistent database into an inconsistent one. Concurrent update processes have the potential for destroying the integrity of a database. The transaction concept is a logical framework in which we can discuss methods to assure consistency during an update.

Murphy's law: anything that can go wrong will go wrong.

Designing a database transaction system requires imagination of what are all the possible ways a system or the people using it can fail such that the integrity of the database is threatened. Then we must design methods that guard against these mishaps. This will never achieve a hundred percent security, but an acceptable level of security with acceptable cost. More security has a higher cost and there is somewhere a balance between what is achieved and what it costs.

4.1 DEFINITION

The transaction concept postulates:

- All changes to a database occur in a transaction,
- A transaction transforms the database from a consistent state to another consistent state.
- The database is initially in a consistent state.

With these rules a database remains always in a consistent state.

> *The initial database is assumed consistent.*
> *A transaction changes the database from a consistent state to another consistent state.*
> *All changes are in transactions.*
> *The database is always consistent.*

The transaction concept is crucial to maintain the database usable over long periods of time. The transaction concept is the framework in which all possible problems that result form the

interaction of multiple updates in an interactive, multi-user environment are resolved.

4.2 TRANSACTION PHASES

A database transaction is started by a process that intends to update the database. A series of retrievals and updates to the database are performed. Finally the process request termination of the transaction—either asking to *commit* the changes to the permanent record or to *abort* the transaction and to delete all the changes. The database then confirms the end of the transaction, either asserting that the changes were committed, or indicating that a failure occurred and the changes could not be retained and the transaction was aborted by the database management system. If the transaction is aborted either by the user or the DBMS, no change to the database occurs.

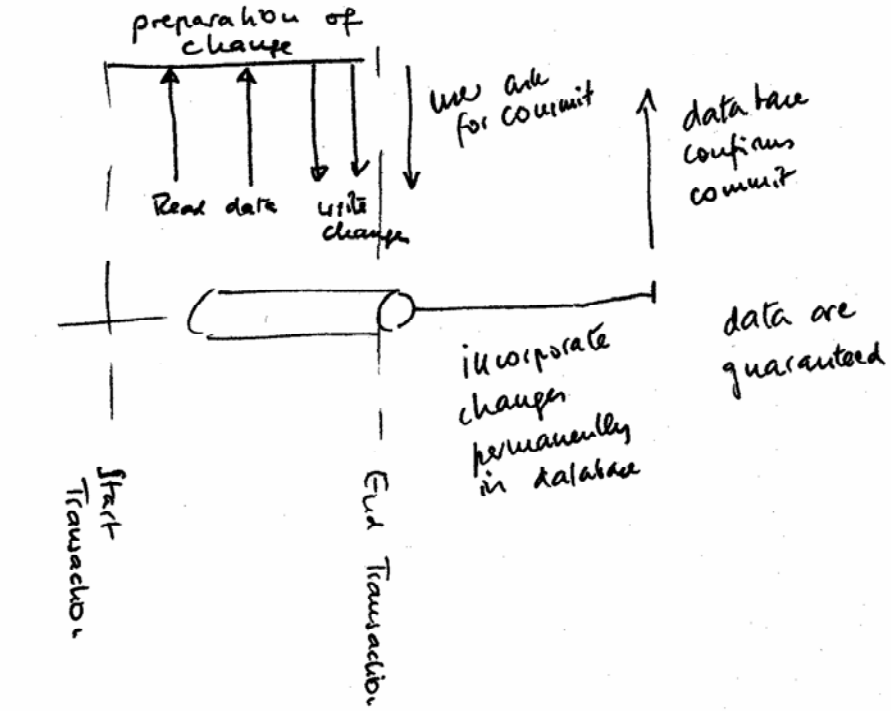


Figure 253: Phases of a Transaction

5. ACID: THE FOUR ASPECTS OF TRANSACTION PROCESSING

A transaction has four properties:

A Atomicity: transactions are atomic operations.
C Consistency; any transaction must leave the DB in a consistent state.
I Isolation: Concurrently applied changes must not interact.
D Durability: If a transaction has completed, then the result of the transaction must never be lost!

This gives the mnemonic ACID.

5.1 ATOMICITY

Atomicity means the logical isolation and indivisibility of concurrent operations. Each transaction transforms the database from a consistent state to a next consistent state. The transaction is completely done or nothing of the transaction is executed.

Definition: A transaction is either completely done or not at all.

```
db2 = doTransaction args db1
doTransaction args db = if consistent db' then db'
else db
     where     db' = changeDB args db
```

How to achieve atomicity? By creating a copy of all the changed data and then replacing the pointer to the original data with a pointer to the new data (Figure 254). It is assumed that changing a single pointer value in the database is atomic—it cannot be done half (even if electric power fails in the moment of changing the pointer).

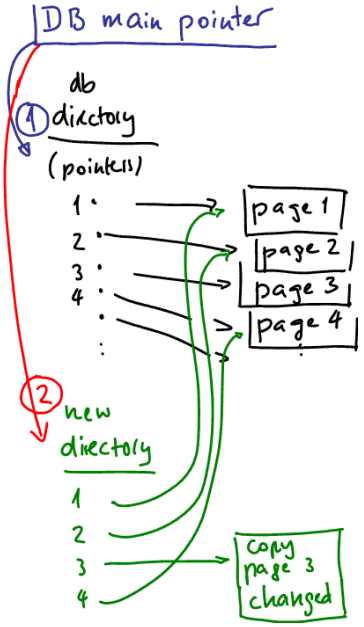


Figure 254: Atomicity achieved by changing only the main pointer

5.2 CONSISTENCY CONSTRAINTS

At the end of the transaction, the database is checking the new state of the database against the consistency rules stated. These rules are expressed as logical constraints on the data stored and will be discussed in the next chapter. Most relational databases allow only a limited set of checks.

5.3 ISOLATION

Concept: guard against unintended interference and contamination between programs; correct execution is equivalent to serial execution of all committed transaction.

If the database is changed by multiple users at about the same time, we must avoid all interactions between the changes of these users: the resulting database must be in the state it was, after the different transactions had been processed sequentially (i.e., one after the other). Any sequence is acceptable, but it must be a sequence, not an *interference* (Bachman 1973, 277) between two concurrent changes. It may well be that the result of the sequence of *f* before *g* is different from *f* after *g* ($f.g \neq g.f$), but both are acceptable solutions for a transaction management.

5.3.1 Danger of concurrent processes

An example from banking demonstrates the need for transactions in concurrent update situations:

```
Three accounts, A has $100; B has $100, C has $100
Two concurrent processes: 1. Put $20 from A to B
                          2. put $50 from B to C.
Consistency constraints: the sum in all three accounts
is always $300.
```

Concurrent processes have the potential to interfere, when one process reads data that the other process has read before and will change later. With the execution shown in Figure 255 the

clients lose money without justification: the three accounts together contain only $280. A correct solution is one, which is obtained by sequential processing of the two requests; in this case, the result is the same, independent of the order of these two processes. This problem is not restricted to banking, but can occur in GIS as well (Figure 260)!

Figure 255: Two concurrent update processes

5.3.2 Concurrency of read and update processes

Transaction management is also necessary to protect programs that only read data if other concurrent users change the data. The rule is that no intermediate state of a transaction can be observed by another user; during a read transaction, the data seen must be from a single state of the database and not changed during the read transaction by another update transaction, and no data written by an aborted transaction must become visible to other processes. Bachman called this *contamination(Bachman 1973)*.

Example of contamination: Process A records that Mr. Smith has moved from X to Y, both communes in county Clare. At the same time process B sums the population of all communes in county Clare. If the process B is not protected by transaction processing to see only a single consistent image of the database, the count can be wrong by one person, either counting Mr. Smith twice or never.

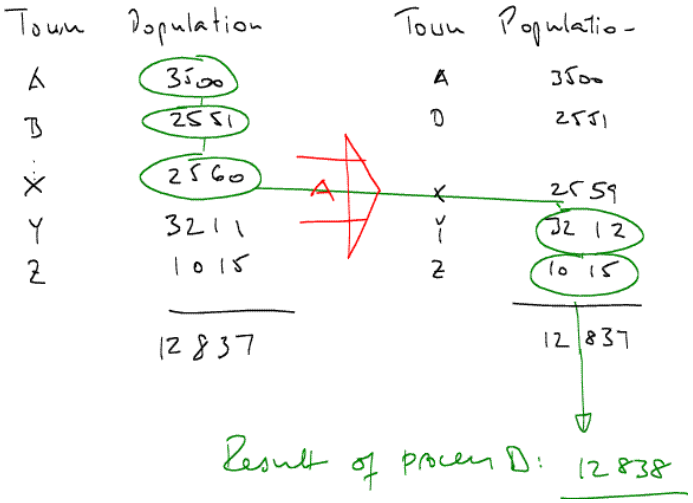


Figure 256: Interaction of two concurrent processes

5.3.3 Concurrency control

Do we conclude from the example that *all* transactions must be executed sequentially? This is a safe solution, but not optimal and not acceptable in today's world. Could one imagine that the large databases that hold all the flight reservations for an airline could only be updated one request at a time? Consider again an

example from banking: if we have to transfer money from M to N and from P to Q, the two processes can read and write in any order without causing problems. Technically we say that the read and write sets of the two processes are disjoint.

The examples above demonstrate that problems occur when two concurrent transactions access the same data elements. Technically, we consider the set of data elements read and written by the transactions (the so-called *read set* and the *write set* of a transaction). Two transactions can progress concurrently if their read and write set does not intersect (Gray, Reuter 1993).

Two different strategies to avoid interference between concurrent transactions are known:

Locking: a transaction locks any piece of data it reads or intends to write; another transaction cannot access a locked piece of data and must wait till the first transaction has concluded and released all locks. With a two-phase locking protocol, all locks must be obtained before any unlocking happens; this is achieved by releasing all locks at the end of the transaction. It guarantees that concurrent processes remain isolated, but cannot prevent dead-lock(Ullman 1982; Haerder and Reuter 1983; Gray and Reuter 1993).

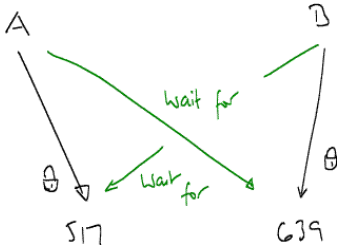


Figure 257: Deadlock: A waits for B and B waits for A

Optimistic strategy: all transactions are permitted to proceed and at the end of a transaction we check if any of the items the process wants to write has been written by a concurrent transaction since it was read by the first one; if this is detected, the transaction is aborted and starts with reading the now current values, otherwise it can commit.

In the presence of updating processes, transaction management is necessary even for processes that only read data.

The two strategies allow the same amount of concurrency and have in the general case the same cost. With the locking strategy there is a potential for a deadlock: process A waits for a lock that process B currently holds; but B waits to obtain a lock, which A currently has—a deadlock has occurred and none of the two processes can advance further. A database may check occasionally for such deadlocks and abort one of the two processes to break the deadlock.

5.4 DURABILITY

Data can be lost when computers or the storage medium fail ("disk crash") and stored values are lost. With Input-Processing-Output processing, data security was achieved by making copies of the inputs, which permits to repeat the processing later again. This is not possible with databases and interactive computing in general. With the interactive computing paradigm, the input data is not available for reprocessing a second time. It is necessary to find a way to assure that data that were entered and the transaction committed to the database is not lost by accident.

The durability rule states that the changes committed to a database must not be lost ever. A naïve answer would be to copy the database before each transaction to another medium, such that it cannot be destroyed all at once. This is not possible, because copying a complete database takes more time than is available between transactions.

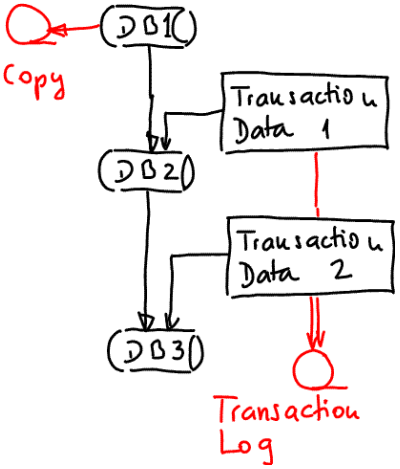


Figure 258: Two Transaction against a database

It is sufficient, to have a copy of the initial database and then preserve the changes applied to it over time. Assume that a copy of the database was produced Jan. 1 (Figure 258). During every transaction, all the changed values are written to a file before they are applied to the database. In this file, the state of a database part is saved in the state it was after the update (so-called 'after images'). This file is stored off-line (e.g., on a magnetic tape)(CODASYL 1971; Gray and Reuter 1993).

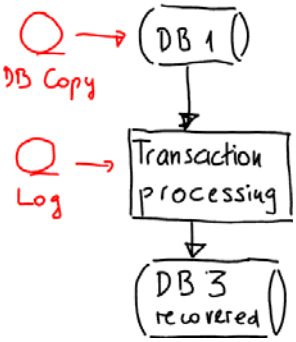


Figure 259: Database recovery

Recovery is possible: assume the third state of the database is lost, but the copy of the database in state 1 is available from the archive (Figure 259). The changes of transaction 1 and 2 that are stored in another file are then played against the database and step by step, the state of the database after transaction 1 and 2 reconstructed.

The same method can be used to reconstruct a previous database state from the current one. This is called to roll-back the database to the beginning of a transaction: before a part of the database is changed, the state it had before the change is saved in a file (so-called 'before images').

Arbitrary high level of security can be achieved, but never hundred percent. The more security, the more cost. Compare the risk of a threat with the cost to guard against it:

- Disk crash—recover the database from backup magnetic tapes;

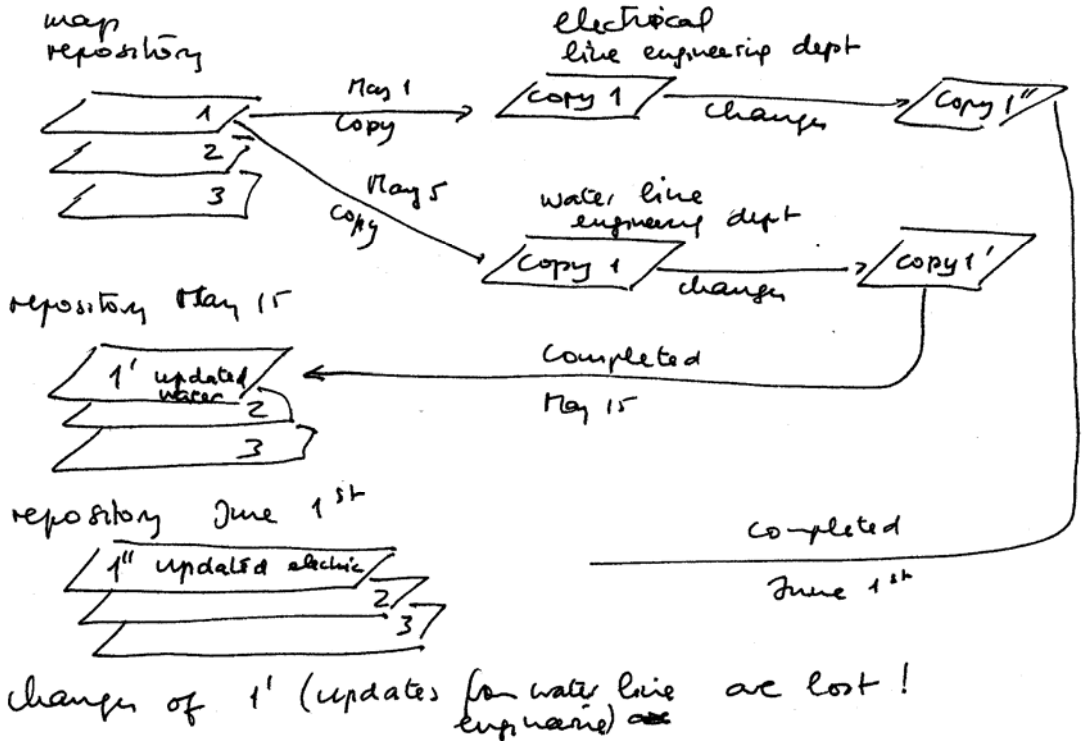
- Fire in the computer room—recover the database from magnetic tapes stored in a secure storage vault outside of the computer room;
- Burning down of building—recover the database from tapes stored in secure storage system at a different location.

One can see higher security against loss of data requires more and more effort; the more devastating an accident is, the less likely the threat usually is, but the more costly the method to secure against it. Truly dangerous and difficult to prevent is human error—either due to incompetence of personnel or even evil intentions of e.g., disgruntled employees.

6. LONG TRANSACTIONS IN GIS

Discussions and implementations of transaction management assume that transactions are short, that is, complete within seconds or minutes. Prototypical examples are reservation systems. Conflicts are resolved or at least detected and one user is made to wait till the other has finished his changes. This is acceptable in many administrative and commercial processing environments: when a client requests a reservation for a seat in an opera performance and the same seat is sold in a concurrent transaction while the client is making up his mind, then the transaction can be stopped (aborted) and a new transaction for another seat can be initiated and the client informed that the seat available a few seconds ago has been sold in the meantime to somebody else.

This does not work for GIS: Assume a collection of maps about public utilities in a city: There is a map for each street, including the water and the electricity lines. Updates are not randomly distributed (as one may assume in an administrative context) in space. Actions in the real world are correlated: constructing a new building will require changes to the electricity, the water, the sewer, and the telephone lines—all in the same street with possible conflicts and concurrency problems. Consider the following sequence of actions: The electricity group requests a copy of the map for the Bräuhausgasse to perform some changes. The water group requests a copy of the same street to update the water lines. The changed map from the electricity group is copied back into the archive and a little later the changed map from the water group is



also copied back—effectively wiping out the changes entered by the electricity group (Figure 260).

Figure 260: Updates are lost because no concurrency control

In GIS—but also in other applications—transactions may be complex and require substantial preparation. It is not acceptable to demand that such work is started again all over. Transaction of this kind last too long for other users to wait for their completion before they can access the same data.

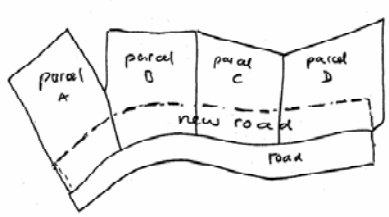


Figure 261: A Transaction locking many entities

Consider a complex transaction of parcels: a road is widened and all parcels on this side of the road must contribute under eminent domain laws a strip of land to the widening of the road. This can be seen as one big transaction including all the parcels along the road and the road parcel as well (Figure 261): In a case in Schlieren (Switzerland), the road was several kilometers long and included literally hundreds of parcels; the transaction was pending for several years due to court cases. Such a transaction requires substantial preparation for the geometric situation and may be delayed for years. Neither ‘restarting’ the procedure nor locking all the parcels involved is acceptable. More intelligent schemes of transaction management must be found at the level of the application domain. In software engineering, similar problems are solved with concepts of versions and branching of development streams, which are merged later.

7. GRANULARITY OF TRANSACTIONS AND PERFORMANCE

The transaction management has serious impact on the performance of a database. When assessing a DBMS it is customary to test functionality and observe the speed with which some operations are executed. It is advisable to check that transaction management is switched on; vendors will 'forget' this, because a DBMS without transaction management runs much faster (twice the performance or better).

Transaction management—especially the concurrency control—can be established at different levels of *granularity*. The simplest solution is to use the full database as the unit of interaction: all transactions interact and must be executed serially; this gives least concurrency and the simplest implementation. The most difficult solution is to select a field in a record or a single entry in a relation as the unit of transaction management: only few transactions will be in conflict, because it is unlikely that two programs need to change the same data field in the same record at the same time; many concurrent actions are possible—but the cost for this fine granularity transaction management in terms of performance may be higher than what is achieved. Effective solutions are selecting physical storage units (disk pages or multiple disk pages) that can be read and written to disk as the units of granularity.

8. SUMMARY

Knowledge is a resource in today's enterprises. Data is centralized to make it available to many parts of the organization. If many users interact at the same time with the data, safeguards must be in place to avoid negative interferences between concurrent updates. The transaction concept achieves this. It consists of four parts:

A Atomicity: transactions are done completely or not at all.
C Consistency; any transaction must leave the DB in a consistent state.
I Isolation: Concurrently applied changes must not interact.
D Durability: If a transaction has completed, then the result must never be lost!

Atomicity of transactions excludes intermediate states of a transaction to become ever visible to any user except the one executing the transaction. The database visible to other processes is in a state of consistency achieved at the end of the transaction.

Intermediate states, which are inconsistent, cannot be completely avoided, but these must never be visible to another transaction.

REVIEW QUESTIONS

- What are the four parts of the transaction concept?
- What is the definition of concurrency? Why is it so detailed?
- How to achieve long term usability of a DB using the transaction concept?
- Explain an example, in which data is lost by incorrect concurrent processing.
- What is interference between transactions? What is contamination?
- What is a correct execution of several concurrent transactions?
- What is the difference between an optimistic and a locking strategy for concurrency?
- What is excluded by the atomicity principle?
- How is durability achieved?
- Why is a transaction mechanism necessary for readers (in the presence of concurrent users who change the data)?
- When does interactive data processing require a database?
- Explain for each of the four components of transaction concept what they prohibit. What is not allowed to happen?

Chapter 18 CONSISTENCY

Databases are created to maintain data useful. The database can check consistency after each update and abort updates that would lead to inconsistent states. In this chapter we discuss the methods to express the consistency constraints. The simplest and most effective method to achieve consistent data is to reduce redundancy.

Redundancy breeds inconsistency!

Consistency can only be discussed in a formal framework. It needs a data model and the rules that are implied by it. Using the relation data model introduced (chapter 16), consistency rules are set in the general framework of logic (chapter 4). Consistency means that the data together with the formalizable rules about the world are not containing contradictions. The expressive power of the language used for the description of the consistency rules determines how much or how little of world semantics can be carried over into database: What rules can be stated and checked at the end of the transaction?

1. INTRODUCTION

A data collection is only useful if the deduced information is *correct*, that is, the isomorphism between the real world and the model world in the information system obtains. This cannot be demanded and checked within the formal framework assumed for the discussion of formal systems (see chapter 3).

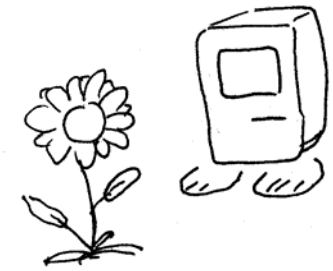


Figure 262: The Banana Jr. Computer checks for correctness of color of flower

Within the context of the information system, we can only check that the data stored is *consistent*, that is, free of contradiction (see 022xx). The integration of methods to assure consistency in a database allows to detect errors during data entry and to correct them immediately.

The importance of consistency in databases has been one of the driving forces behind the development of the field. In the early days numerous ad hoc attempts were made to identify practical rules useful for the design of database schemas. It was driven originally by a hope that consistency of data could be described by syntactic rules. It culminated in the collections of rules about Normal Forms, from 2nd to 4th, 5th, and higher Normal Forms (Ullman 1982; Date 1983)

Parallel to the development of relational database theory by Codd and others, investigations into a logical interpretation of a database were pushed. The seminal paper by Gallaire, Minker, and Nicolas (Gallaire 1981; Gallaire, Minker et al. 1984) and the corresponding book (Gallaire 1981) opened a new way to understand a database and the methods to express consistency constraints using logic.

2. THE LOGICAL INTERPRETATION OF A DATABASE

Standard database theory gives the semantics of the operations in terms of algorithms, which deduce values from a given database by a search method. Codd has shown that relational theory is *database complete*; all facts stored in the database can be retrieved with the operations given. This is—from a mathematical point of view—a model: the operations are explained in terms of their effects on a model—possibly a very simplified model of it, represented in computer storage (chapter 4xx).

A logic view considers the database as a set of facts and a query as a proof: demonstrate that the query result follows from the stored facts. Gallaire, Minker, and Nicolas (1984) have pointed out that searching a database is like doing a logical proof. The database can be seen as a set of axioms—which is an extensional definition of relations—and the query as a proof. The query can be a question ‘what *x* fulfills the properties *p*’ and the result gives a value for *x* (see example of backward chaining in chapter 4xx).

The logical framework is more general than a data model with its corresponding algorithms for computing the result of a query. The relation framework is equally powerful to the logical framework. Bird and deMoor (1997) show that for unitary, tabular allegories (as used above in chapter 16), everything that can be proven in a set theoretic framework is also true for relations.

The framework of logic and the concept of "query as a proof" allow the classification of different collections of knowledge. Relational databases have a simple structure, namely collections of tuples that describe facts (which means Horn clauses with *m=1* and *n=0*, see chapter 4). In such systems, proof reduces to search. A trivial algorithm to find an answer is to start with the first element and to check this and any following

one till one reaches the end—the answer to the query is then 'there is no such element' — or till a tuple is found that fulfills the condition. More performant algorithms are just faster arriving at the same result.

The logical interpretation of the database is promising, as it helps to discover:

- What are the logical rules assumed and built into the database without being stated?
- The expressive power of the database: what kind of facts can be expressed in the database and specifically what cannot be expressed?

3. LOGICAL ASSUMPTIONS WHEN QUERYING A DATABASE

With the adoption of a logical viewpoint, database theory can be compared to logic. In particular, one can ask what the deduction rules are and what axioms are implied in query processing. Reiter has identified a number of assumptions, which are automatically and tacitly made in relational data processing(Reiter 1984). The implied rules in databases follow from the ontological commitments appropriate for administration, but they are not always applicable for information systems about physical reality.

There are three assumptions invoked:

- the closed world assumption says, that we know all what is there and what we do not know is false;
- The domain closure assumption says that all the individuals are known; and
- The unique name assumption says that distinct names relate to distinct individuals.

3.1 CLOSED WORLD ASSUMPTION

The assumption that all facts about the world are known allows in database query processing concluding from the absence of facts that something is not true. This rule is known as 'negation as failure' (e.g., in the Prolog language(Clocksin and Mellish 1981; Colmerauer, Kanoui et al. 1983)): the negation of a fact is expressed by its absence, and thus by failing when one searches for it without success. Negated facts are not stored explicitly—which is effective: Consider how many things are not true in the world and how much storage space would be needed!

Closed Word Assumption:
What we do not know is false!

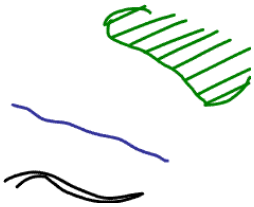


Figure 263: A map with a wide meadow between a road and a forest

Figure 264: The same area in reality, the forest has grown to the brook and a building was constructed

Unique Name Assumption:
All entities have one and only one name.

This is effective in administration, where the database is the ultimate arbiter on questions like 'is Z a client of this bank' or 'is A student at UCSB'. If Z or A are not in the respective database, they are not a client or student there!

For a GIS database, this is not as simple: we have never complete knowledge of the world, thus from an absence of knowledge one must only conclude 'we do not know f', but not 'f is not the case'. If a piece of land does not show a building, we should not conclude that there is none. We only know that there was none of the kind considered relevant when the data was collected. Land on a map without trees is at best a statement that no trees were on the land when it was surveyed, but one must not conclude that this land is currently not tree covered—trees could have grown since (Figure 263 and Figure 264).

The use of the closed world assumption in a GIS must be selective and each relation should be labeled for completeness, thus indicating if absence can be interpreted as negation.

3.2 Unique name assumption

The relational database query methods assume that all individuals have unique names. One can thus conclude that one name always means the same person and those two individuals with different names are not the same.

Again, this is a dangerous assumption—even in administrative processing. For example, I had once a student, Kevin L. Johnson, who had the exact same first name, middle initial, and last name as another student at the University of Maine. The other Kevin L. Johnson student was dismissed, because he had failed some courses and our student found himself dismissed, because for the administration, the two were only one. The billing however seemed to have worked independently and both paid their tuition…

In a GIS, we may have the situation, that the same object is entered with two different names (Milan and Milano for the northern Italian city), or two times the same name appears, but describes different things—Calais, Maine and Calais, France—just the pronunciation is different!

3.3 Domain closure assumption

Domain closure states that all the individuals that exist in the world—and could appear in the proof—are known in the

database and no other individuals exist. Unless we assume domain closure, we could never answer questions like: find all cities with more than 100'000 inhabitants in Antarctica. The response is, of course 'none', but only if we assume that we have a complete inventory of all cities in Antarctica. Or, even more tricky: ask whether two individuals ever went to the same university; if we cannot assume that we have a complete list of all universities, there could exist one, of which we do not know anything and the two individuals both went there.

Closed Domain Assumption: All individuals are known.

4. INFORMATION SYSTEM: A DATABASE PLUS RULES

In an information system, the database is augmented with rules (chapter 15). If the rules cannot be expressed in a form that can be stored in the database, then the rules are included in the application programs, but not in a format that makes it easy to see what the rules are and where they are expressed. Rules in programs are not used by the transaction management to check consistency at the end of all transactions.

The database schema must contain as many of the consistency rules as possible—initially, it was hoped that all the consistency rules can be expressed in the form proposed in database schema languages. This is not possible and will not be possible, unless the language to express the constraints has full computational power (i.e., a full programming language).

The difficulty is aggravated by interaction of rules between transactions: assume there is a constraint stating that either A in table X or B in table Y exists. In a routine to introduce A we check for the absence of B in table Y—but how can we lock the absence in table Y? It is possible, that a concurrent process is inserting a B in Y while the transaction to insert A in X is underway and the conflict is not detectable (unless the first process locks all of table Y).

5. REDUNDANCY

Data is stored redundantly if it is stored repeatedly. Redundancy is desirable to guard against data loss: we archive copies of the database. The database however should not contain duplications, because duplication permits contradictions: if a fact is stored twice it is possible that only one of the two copies is updated and then they have different values, which is a contradiction.

Redundancy is a more subtle concept than just duplication of storage: a logical system contains redundant clauses if we can delete a clause and still derive all the same conclusions as we could from the whole system (compare the discussion of Euclid's five axioms, chapter 7). In logic, we say that the clauses are independent; dependent clauses indicate some form of redundancy. Of course, if clauses are not independent, changing one without the other can create an inconsistency.

Famous example: The proof that the fifth (parallel) axiom in Euclid's elements is independent of the others.

Consistency considers data and rules. Even data that does not duplicate directly the same measurements can contain redundancy and inconsistencies. Take a simple case of storing the noon temperature of a day for several cities of the world in a table; to accommodate different cultures, we store temperatures in °*C* and °*F*:

City	*Temperature* ℃	*Temperature* ℉
New York	35	95
Berlin	29	84
Vienna	26	79
Rome	32	32

Observe the contradiction in the temperature for Rome (correct 90°F!

The table itself has no redundancy, but with the knowledge of a conversion formula from Centigrade to Fahrenheit (see chapter 6) redundancy becomes obvious: one of the two temperature columns is superfluous and can be deleted and reconstructed when needed using the conversion function.

6. EXPRESSIVE POWER

In a logic view, one can investigate the expressive power of a database and the rules it allows. The most general case of a proof system accepts arbitrary collection of first order formulae and deduces the result like a mathematical proof. No effective method is known so far to find automatically a proof for the most general logical system. The simplest case is a set of facts and only simple queries that can be answered with a sequential search. A wide spectrum of expressive power and performance is available (see figure 300-02 in chapter 4).

Relational database allows only ground facts; it does not allow rules or negated facts ("Peter does not live in Vienna"). This is a limitation in the expressive power of the storage of facts. Another shortcoming of relational databases query languages is that they are not computationally complete. This means that there are things that can be computed, but cannot be computed with relational algebra. In relational algebra recursion

is missing; this seems not of much use in administration—except for the processing of bills of components, which have components themselves. It is however necessary for GIS, where operations that require closure are common:

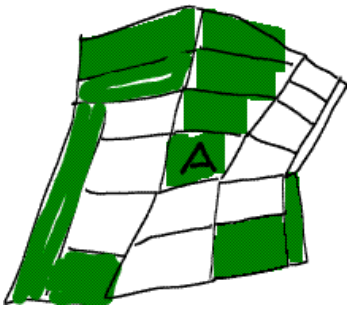


Figure 265: Find all connected wood parcels

Example: Find the connected wood area, given a set of plots (some wooded) and their neighborhood connections (Figure 265). Starting from a given wooded plot, say A find all connected wooded plots and then all the connected ones to those, etc., till no more are found (this is called the fixed point: $f\,a = a$). This is not the same result as to find all the wooded area in Figure 265, which would be a zone (see chapter 14xx)!

7. CONSISTENCY VS. PLAUSIBILITY RULES

Databases add rules to check the *plausibility* of the data: the age of a person cannot be more than 100 years, the number of stories of a building must be less than 100, a birth year must be in the range of 1900 till 1999, etc.

As the last example shows, for all these plausibility rules, exceptions are possible. Plausibility rules are useful to check data and ask for confirmation if values outside the plausible range are entered, but they must not make it impossible to enter such values, e.g., my grandmother was 101 when she died.

8. SUMMARY

A database is consistent, if the collection of data and the rules are logically consistent. The framework of logic applied to databases reduces database consistency to logical consistency—and makes clear, that consistency is only meaningful for rules together with the data. Most database management programs do not provide a language expressive enough to capture all consistency rules and these are hidden in application programs.

8.1 REDUNDANCY BREEDS INCONSISTENCY

Redundant storage means to store something twice; and then, the two copies can differ. The redundancy can be in the data or result from the combination of data and rules, which make it possible to construct stored data or deduce data in more than one way. Redundancy is to be avoided, not because it wastes storage, but because it can lead to inconsistence.

8.2 *Redundancy depends on rules*

The observation whether some facts are redundantly stored or not depends on the rules and the facts, not the facts only. If there is a rule that says that the relation between ZIP code and name of town is a function (i.e., a simple and entire relation—see chapter 16), then the relation table in the previous chapter contains redundancy.

The difficulty is that the world is not simply cut and does not follow the rules we make to simplify our conceptualization. There are few rules that have no exceptions.

The world is more complex than examples in (database) text books.

8.3 *Expressiveness of data model and query languge*

Data models restrict what facts can be included in a database; the relation (and the relational) data model restrict facts to positive statements. Negative knowledge cannot be inserted in the database.

Query languages like SQL are not computationally complete; they lack recursion or a fix point operation, which hinders computations that need some form of closure. If the query language is also used to express consistency constraints, the same limitation applies there.

Review Questions

- Explain Functional Dependency? What is different in a Multivalued Dependency? Give examples for both.
- Why is redundancy considered harmful?
- What is lossless decomposition?
- What are the logical assumptions built into the query strategy of a relational database? What is the Closed World Assumption? Why is a domain closure assumption necessary?
- What is meant by ‘negation by failure’?
- How to represent negative facts in a database? Give an example for a negative fact?
- Why and how can a query be viewed as a proof in a logical system?

PART SIX GEOMETRIC OBJECTS

Part three introduced coordinates to represent point locations. Our conceptualization of the world uses complex geometric objects: parcels are defined by corner points that are connected by straight lines. In this part, a first step towards the representations for geometric objects is made, namely the representation of straight lines and similar infinite geometric objects. The part consists of two chapters only: the first discusses straight lines in *2d* space and presents a solution for the calculation of the intersection. The second chapter then generalizes the approach *n* dimensions and *m*-dimensional geometric objects.

A corresponding discussion of temporal objects is not needed: there is only a single infinite time. Only bounded temporal objects, namely intervals, are of interest. These will be discussed in part 6 together wit the corresponding bounded spatial objects.

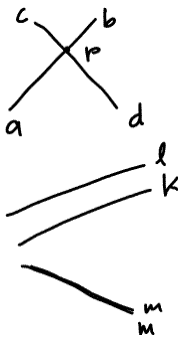


Figure 266: (a) Simple case for line intersection, (b) parallel lines l and k do not intersect, and (c) collinear lines m and n have infinitely many intersection points

The computation of intersection of two lines is an example why geometric computations are complicated: besides the simple case where the two lines intersect, there are numerous special configurations that do not lead to a solution. The two lines can be parallel or even collinear (Figure 266). The approach used here—using homogeneous coordinates and projective geometry (see chapter 10)—leads to an operation that is total, i.e., produces a meaningful result for all inputs. The same approach then gives also a dimension independent solution for the general case—the intersection of planes with lines, planes with planes, etc. This is one of the building blocks from which a GIS is constructed. The application of the theory in this chapter are methods to construct geometric objects, as included in CAD and GIS programs(Kuhn 1989). For example, construct a parcel with a boundary parallel to another one of 15 m width from a given one, which requires the computation of numerous line intersections (Figure 267).

Line intersection p of two lines given by 4 points:
p = (a x b) x (c x d)

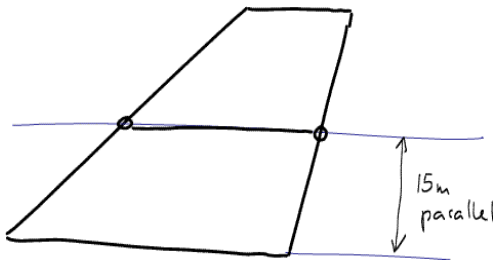


Figure 267: Geometric construction with conditions

Chapter 19 DUALITY IN PROJECTIVE SPACE: INFINITE GEOMETRIC LINES IN *2D*

Points can be represented and stored in the database (see part 5), but we have not yet seen how to represent lines and higher dimensional infinite geometric objects. In this chapter a representation for infinite straight lines is presented that are the geometric objects most often used to delimit spatial objects.

Straight lines in the plane and their intersection are studied since the Greeks investigated geometry(Heath 1981). The analytical solution for intersection of two linear equations works only for the 'normal' case of lines that have real intersection points. Computations must separate the treatment of special cases, e.g. parallel lines, because computations with parallel lines lead in many formulae to divisions by zero

Homogenous coordinates were introduced to achieve a general solution for linear transformations. They are founded in projective geometry where all lines intersect (no special case for parallel lines)! Using the same embedding of the plane into the projective space we used to generalize transformations (see chapter 10) a formula which works for all situations is found. This chapter concentrates on straight lines in *2-dimensional* space; the next chapter will discuss the general case in *3*- and higher dimensional spaces.

Note: Duality links a primal space to a dual space.

This chapter is using duality. Duality (introduced in chapter 5.3xx) is a method to reduce the number of axioms, theorems, and proofs by half: statements with the two terms X and Y remain true when systematically all terms X are exchanged for Y and all Y for X(Stolfi 1991). Duality is based on a morphism (duomorphism) and can even be used for implementation(Guibas and Stolfi 1987).

We have encountered duality before:

- Boolean algebra is dual: the axioms of Boolean algebra are valid when one exchanges systematically *and* for *or* and *T* for *F*.
- A set partially ordered by ≥ is dual to the same set partially ordered by≤.

Similarity in symbols:
∪-> v, ∩-> ∧

- In set theory we can exchange the *union* and *intersection* operation and exchange the *null set* against the *all set*.
- For lattices join and meet are dual.
- The duality between a right and a left module, which we have encountered in chapter 10xx discussing the construction of a vector space as a (right) module.
- A category and the category with all the arrows reversed (the opposite category) is dual; this was used when discussing relations (in chapter 16xx).

1. REPRESENTATION OF LINES

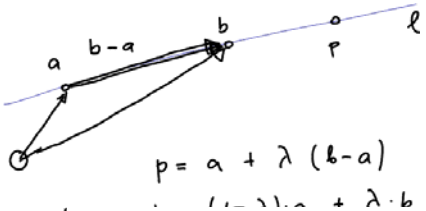
Assume that points are stored in relations *point :: ID -> coord*, where *coord* are 2 tuples (general case *n*-tuples) from *R x R*. What is a suitable representation for infinite lines? Several methods are used to suit particular applications.

1.1 VECTORS

A line given by two points *p*, *q* can be represented in vector notation as (Figure 268):

$p = a + \lambda \, . \, v = a + \lambda \, . \, (b - a) = a \, (1-\lambda) + b \, (\lambda).$

The last form can be read as a weighted mean from the two points. The above formulae in vectors can be separated in corresponding formulae for *x* and *y* coordinates.



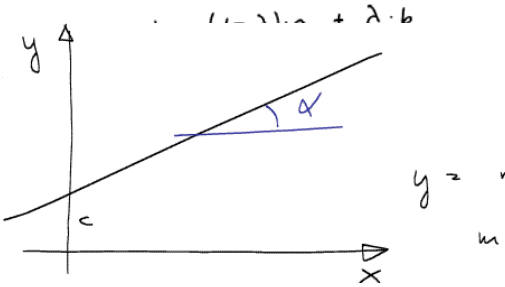


*Figure 269: y = m * x + c*

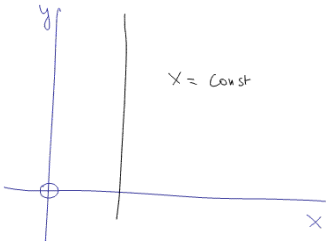


Figure 270: Line parallel to y axis

1.2 FUNCTION

In coordinate space, the most often used representation is

$y = m * x + c$, *(Figure 269)*

but this cannot represent lines parallel to the *y* axis (Figure 270).

1.3 NORMAL TO THE LINE

For lines in the plane, a line is the locus of all vectors from a given point and orthogonal to a vector on the line (Hesse Normal form). This gives the representation (with Φ the direction of the normal on the line):

$x \cos \Phi + y \sin \Phi - d = 0$ where $\Phi = v + \pi/2 = b\text{-}a + \pi/2.$

This can be generalized to a representation of any line in the *2d* plane by three homogenous values *t,u,v*: $t\,x + u\,y + v = 0$. This representation is homogenous, because the line

$\lambda * t * x + \lambda * u * y + \lambda * v = 0$ $\lambda \neq 0$

is the same line; a line has only two degrees of freedom and a representation with 3 values is necessarily homogenous. The Hesse Normal Form (Figure 271) is the one for which $sqrt\ (a^2 + b^2) = 1$. Because the vector from any point of the line *p* to a

given point *a* of the line must be orthogonal to the vector *n*, all points *p* of the line must fulfill

$n \cdot (p - a) = 0.$

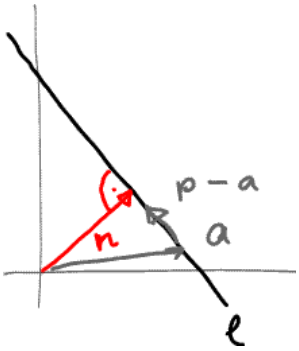


Figure 271: Hesse Normal Form

The equivalence can be used to compute the values for *t, u,* and *v* when two points *a* and *b* of the line are given:

$$v = b - a = (x_b - x_a, y_b - y_a)$$
$$n = (y_a - y_b, x_b - x_a)$$
$$p = (x, y)$$
$$n \cdot (p - a) = 0$$
$$(y_a - y_b)(x - x_a) + (x_b - x_a)(y - y_a) = 0$$
$$x(y_a - y_b) + y(x_b - x_a) - x_a y_a + x_a y_b + x_a y_a - x_b y_a = 0$$
$$t \cdot x + u \cdot y + v = 0$$

$$t = y_a - y_b$$
$$u = x_b - x_a$$
$$v = x_a y_b - x_b y_a$$

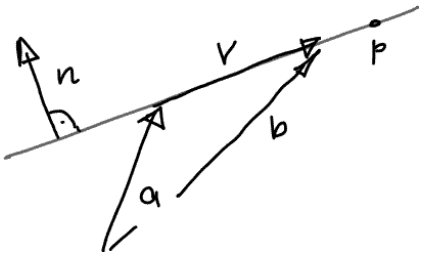


Figure 272: p = n . (b-a)

1.4 LINE DEFINED BY COLLINEARITY

An argument in 3 dimensional space gives a formula for the line *l* such that for any point *p*

$l \cdot p = 0.$

Any point *p* of the line must be collinear with the two given points *a* and *b*, such that *triple (a, b, p) = 0.* From *triple (a, b, p) = (a x b) . p = 0* and *l . p = 0* follows *l = a x b.* The same result is obtained when using homogenous coordinates and the relation *triple (a, b, p) = det [a,b,p]*, which is

Point p on line l
l . p = 0
Line through a b
l = a x b

$$\det \begin{bmatrix} 1 & 1 & 1 \\ x_a & x_b & x \\ y_a & y_b & y \end{bmatrix} = 0$$

By expanding the determinant for the last column we obtain the same values as above:

$$-x \cdot \begin{vmatrix} 1 & 1 \\ y_a & y_b \end{vmatrix} + y \begin{vmatrix} 1 & 1 \\ x_a & x_b \end{vmatrix} + \begin{vmatrix} x_a & x_b \\ y_a & y_b \end{vmatrix} = 0$$
$$x \cdot (y_a - y_b) + y(x_b - x_a) + x_a y_b - y_a x_b = 0$$

2. INTERSECTION OF TWO INFINITE LINES

The computation of the intersection of two straight lines has a closed solution, if we use the homogenous coordinates introduced in chapter 10xx.

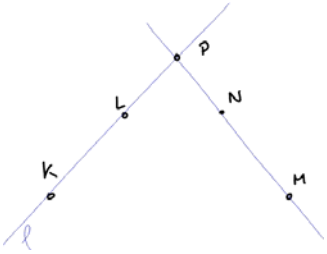


Figure 273: Intersection of two lines

2.1 2D-Analytical Geometry

Given two lines represented as two homogenous equations in two unknown, namely the coordinates of the intersection point $p = (p_x, p_y)$:

$a_{11} * x + a_{12} * y + p_x = 0$
$a_{21} * x + a_{22} * y + p_y = 0$

expressed in vectors and matrices:

$A\,x + c = 0.$

The intersection p (p_x, p_y) is found as the solution of the two simultaneous equations. Using the standard solution formula, this gives:

$x = A^{-1}\,(-c).$

for which Cramer's rule gives the coordinates:

$$x = \frac{\det \begin{bmatrix} c_1 & a_{12} \\ c_2 & a_{22} \end{bmatrix}}{\det A}, \quad y = \frac{\det \begin{bmatrix} a_{11} & c_1 \\ a_{21} & c_2 \end{bmatrix}}{\det A}, \quad A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}$$

A test is necessary to avoid division by zero, i.e. $det\ A = 0$, which occurs when the two lines are parallel and have no intersection point. The function is not total and does not always give an answer.

2.2 Line intersection Computed in homogenous space

In projective space, two lines always intersect. A computation with homogenous coordinates produces always a result. Given two lines m and n, the intersection $p = m \times n$ is obtained from

triple m n m = triple n m n = 0 (triple product with coplanar vectors is always 0!)

we obtain:

$m \cdot (n \times m) = n \cdot (n \times m) = 0,$

replace $(n \times m) = p$ and obtain:

$m \cdot p = n \cdot p = 0.$

This shows that p is a point of the line m and also of the line n, which is the condition for the intersection point.

Intersection point x of two lines m n
p = m x n
If lines m given by points a, b and line n given by c,d then
p = (a x b) x (c x d).

A geometric justification for this result is possible following a model introduced by Menger(Blumenthal and Menger 1970). To represent 2-dimensional space select a point O (for origin) in 3-dimensional space. A 2-d point is represented by the line through the origin and the point, called an O-line. A line through two points is in homogenous space the plane through the origin and the two points (recall: three points define a plane!), which he alls an O-plane. We have used this representation earlier when introducing the general linear transformation in (see section

10.7xx). In Figure 274 two lines *m,* and *n* are given in the *z=1* plane with points for *m (p,q)* and for *n (r s)*.

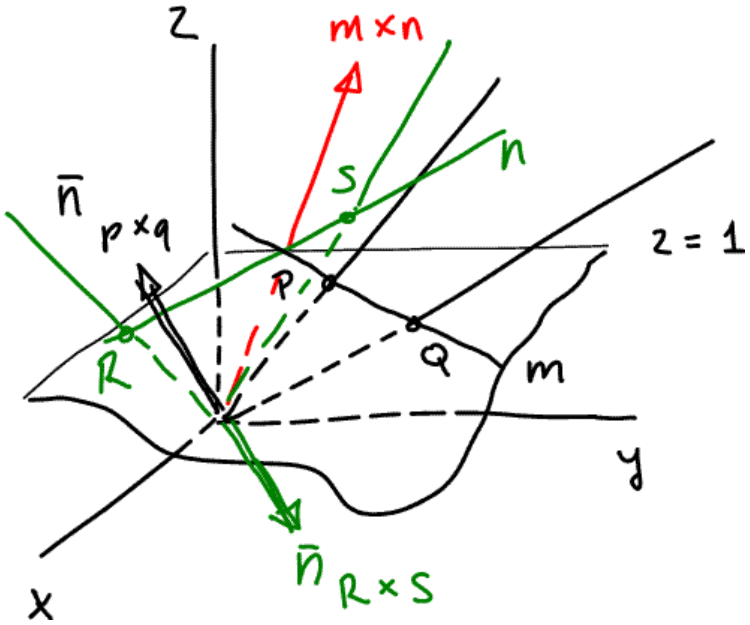


Figure 274: The intersection of two lines—Line in homogenous coordinates

The points *p* and *q* determine a O-plane in the homogenous space, so do *r* and *s*. The intersection point is the O-line through the origin and the intersection point. This is the intersection of the two O-planes determined by *p*, *q*, respective *r, s* and the origin. If we construct the normals to the two planes *m* and *n*, we know that each line in *n* must be normal to the normal of *m* and the same for all lines in *n*. The intersection line is in *m* and *n* and thus normal to *m* and normal to *n*, that is, the normal on the plane given by the two normals.

This leads to the formula expressed in vector notation in *3d* space: For each plane, the cross product gives the normal to the plane. Consider the plane through the two normals (and again through the origin). The normal on this plane is the intersection of the planes from *p, q* and *r, s*, that is, the intersection of the normal with the horizontal plane *z = 1* is the intersection point. This gives the same formula:

v =m x n = (p x q) x (r x s).
m= p x q is line p,q, n= r x s is line r, s

2.3 Special Cases

This formula is a total function; it gives a result, even for parallel lines, but the intersection is not necessarily a real point. The intersection of two parallel points give a result with a homogenous value of 0 and the transformation to the Euclidean

representation is not possible (see formulae in section 3 following).

The vector computation does even produce a result if a line is erroneously defined by twice the same point *p*. The corresponding line l = p x p is the 0 vector (i.e. a vector with all components 0), which is not a geometric element in Menger's model used above.

2.4 LIMITATION TO 2-DIMENSIONAL GEOMETRY

Above we have used the homogenous coordinate space represented as *3d* and followed a Euclidean (*3d*) argument, using vector operations in *3d*. We have used the cross product, which is defined for *3d* vector space only, which limits this formula to the special case of lines in *2d* space, which transform to *3d* homogenous coordinates. Homogenous coordinates are a representation of the projective space. This will be explored in this and the following chapter to arrive at a dimension independent solution.

3. PROJECTIVE GEOMETRY

Homogenous coordinates were popularized in computer graphics to avoid division(Newman and Sproull 1981), but projective geometry can contribute more to computational geometry than just a computational trick to improve performance. Projective geometry is an example of a non-Euclidean geometry, where straight lines always intersect (see chapter 7). It is an example of a functor (chapter 6): a situation where we have not enough elements to represent all situation is solved with a morphism to a representation with more elements. Projective geometry is constructed from the ordinary Euclidean plane, to which a line at infinity is added.

There are different models for projective geometry(Stolfi 1991), which can be used to aid imagination:

- the spherical model,
- the straight model,
- Menger's model of 0-lines and 0-planes, and
- the analytical model;

so far we have used only the analytical model.

3.1 THE SPHERICAL MODEL

The projective plane is the image of a (half) sphere (Figure 275), projected stereographically on a plane (Figure 277). There are no

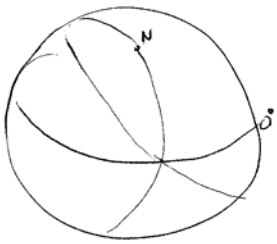


Figure 275: The sphere with geodesic lines

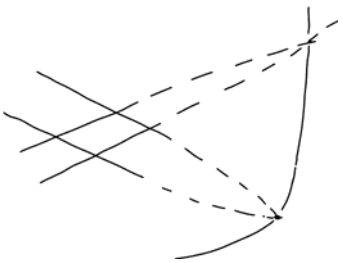


Figure 276: Parallel lines in the projective plane intersect. The intersection point is on the ideal line.

pairs of geodesics that do not intersect: all great circles on a sphere intersect.

3.2 THE STRAIGHT MODEL

The straight model is the projection of the half sphere to a plain touching the sphere in a pole (Figure 277). It contains all the points of plane plus the points of the infinite line, which is the image of the equator (Figure 276).

Stereographic projection maintains collinearity between geodesics. The geodesics of the sphere, the great circles, are mapped to straight lines. As all great circles intersect, all straight lines in the projection intersect as well. Some of the intersection points are on the great circle of the sphere in the equatorial plane, which is projected to the line at infinity. The intersection points go equally to infinity, indicating the lines in the projection are parallel (Figure 276).

3.3 MENGER'S MODEL OF 0-LINES AND 0-PLANES

Menger has suggested a model for an axiomatic treatment of projective geometry, representing points in the projective plane by lines through the origin (O-lines) and lines through planes through the origin (O-planes). This model was used above to justify geometrically the formula for the intersection point. It generalizes for *n* dimensions and translates immediately to the analytical model.

3.4 THE ANALYTICAL MODEL

It consists of the vectors $[w, x_1, x_2, x_3...]$, which are considered as homogenous, that is, they represent the same point when multiplied with any constant $(\neq 0)$. It is the model we have used to represent homogenous coordinates (e.g., in Figure 274). It is also a model which generalizes beyond 2-dimensional geometry.

Note: I order the coordinates such that the homogenous coordinate *w* is the first element in the vector; this prepares for generalization to n dimension. Most authors place the homogenous coordinate *w* as the last element!

w is first coordinate!

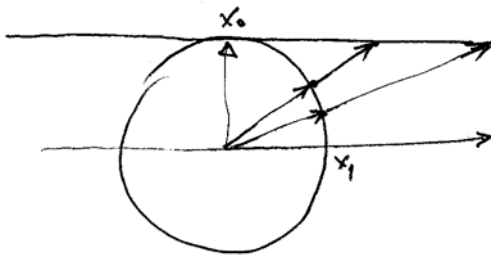


Figure 277: Projection of a half sphere to a plane

3.5 CONNECTION OF THE MODELS

The models are connected by central projection from the origin. Computing with the projective plane is just a different interpretation of the geometric situation and a different representation. The *2d* vectors are transformed to homogenous coordinates of dimension 3 (see xx), where the mapping between the two is:

$x = xh / wh$ $\quad xh = x$
$y = yh / wh$ $\quad yh = y$
$\quad wh = 1$

the mapping to the unit sphere is

$d = sqrt\ (sqr\ w + sqr\ x + sqr\ y)$
$xs = x/d;\quad ys = y/d;\quad ws = w/d.$

This corresponds to a central projection of R^3 onto the unit sphere or onto the plane tangential to the sphere at *(1, 0, 0)*. The additional coordinate *w* can be seen as a scale factor, with which all coordinates are scaled at the end. Figure 278 shows the correspondence between *as* and *a*; it also shows that *a* and *a*, *b* and *b* are the same point, expressed as homogenous coordinates.

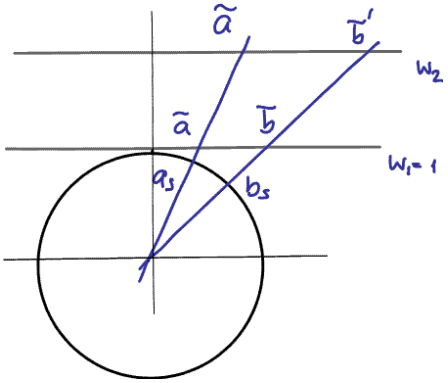


Figure 278: Homogenous coordinates a and a', b and b'

The value of the first coordinate in the homogenous representation indicates, whether the point is a regular point ($w \neq 0$) or is an ideal point at infinity ($w = 0$).

4. DUAL SPACES: FROM POINTS TO FLATS

The observation that the representation of a *line* has the same form than the representation of a *point* in homogenous coordinates suggests a duality between lines and points in projective space. In the projective plane,

"(i) Any two distinct points lie on a unique line;

(ii) Any two distinct lines intersect in a unique point.

The incidence properties (i) and (ii) are dual to each other, in the sense that the interchange of the words "point" and "line", plus a minor change in terminology, changes property (i) into property (ii) and vice versa." (Mac Lane and Birkhoff 1991, 592).

4.1 CONSTRUCTION OF A PROJECTIVE SPACE AS A LINEAR ALGEBRA

To construct the projective plane, take a *3d* vector space *V* over *F* and define the points *P* as the *1*-dimensional subspaces of *V* and the lines *L* as the *2*-dimensional subspaces of *V*. These are the flats, i.e. the O-points and O-lines of Menger's model. This is visualized as in Figure 281: the points are the lines through the

origin, the lines the planes through the origin. Parallel lines have intersection points in the infinite line, hence the popular statement 'parallels meet in the infinite'

For these points and lines and two operations join and meet, the following rules apply:

> "(1) The join as well as the meet of any element X with itself is X.
> (2) If a point and a line are incident, then their join is that line and the meet is that point.
> (3) Besides the points and lines, there exist a flat V (the vacuum) and a flat U (the universe) such that the join of V and any flat X is X, and the meet of V and X is V; and that the meet of U and X is X;, and the join of U and X is U.
> (4) the meet of distinct points is V; the join of distinct lines is U.
> (5) a point and a line that are not incident have the join U and the meet V." (Blumenthal and Menger 1970136)(a comparable description in terms of vector spaces is given in (Mac Lane and Birkhoff 1991, 592)).

These are the axioms of a lattice with units <L, ∧, ∨, U, V> (see chapter 16xx); this is no accident, as historically, work on an algebraic formalization of geometry has produced lattice theory.

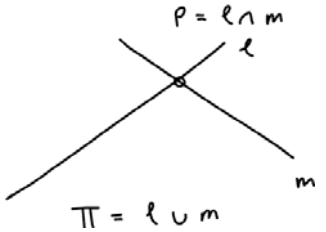


Figure 279: Intersection of two lines is a point

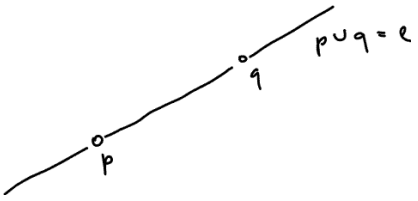


Figure 280: Union of two points is a line

4.2 POINTS AND LINES ARE DUAL

We want duality in *2d* space to have the following properties:

- the dual from a dual space is the original space.
  dual . dual = id.
- the dual of a point is a line, the dual of a line is a point.
- duality preserves incidence: if a point is incident with a line then the dual line is incident with the dual point.

Figure 282 shows a construction of the duality between line and point—it reminds us of the Hesse Normal Form —that is related to the representation of lines as homogenous coordinate triples and as homogenous points (note that there are other forms of duality between points and lines, useful in different contexts).

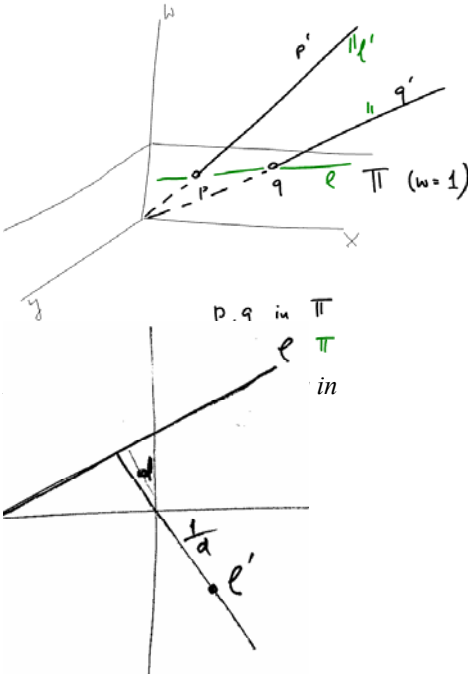


Figure 282: A line l and its dual (the point l')

This construction of duality preserves incidence as can be seen in the following Figure 283. The lines *a* and *b* intersect in point *l*, the dual points *a'* and *b'* are connected by the line *l'*, which is the dual of the point *l*.

Duality can simplify geometric computation: we have the choice to compute in the primal space or in the dual space, whatever is simpler. It is generally simpler to construct a line connecting two points than to compute the intersection. The figure shows, that we can determine the intersection of two lines

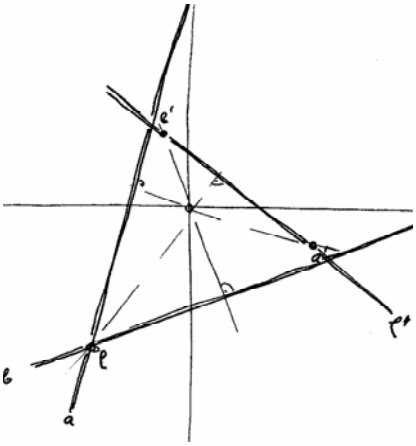


Figure 283: Duality preserves incidence

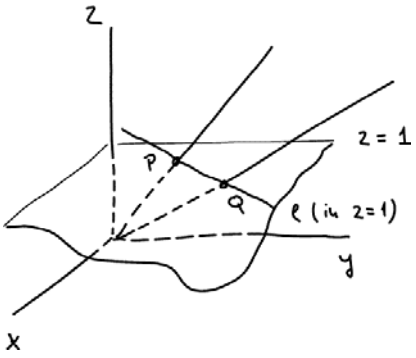


Figure 284: A line is a plane in the homogenous space

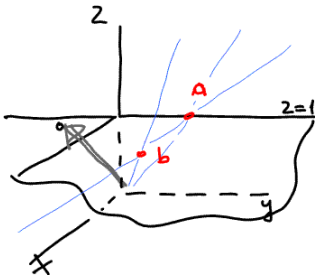


Figure 285 A parallel line and its dual

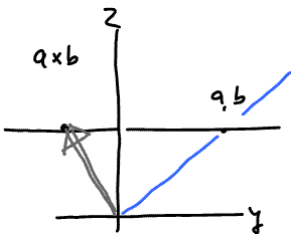


Figure 286: Cross section of figure 18

by connecting the corresponding two dual points and to determine the primal point belonging to this line.

4.3 Duality in Homogenous Space

In Menger's model of the projective space, where a *2d* point is a O-line in *3d* space, which goes through the origin and the *2d* point in the horizontal plane $z = 1$ (see Figure 284), duality can be explained in a visual form:

A line in *2d* (given as 2 points) is a plane in the homogenous space—namely the plane through the origin and the two given points. The dual of this homogenous plane (a *2d* line) is the normal on this plane—a line in homogenous coordinates and correspondingly a point in *2d* (the intersection of the normal with the horizontal plane $z = 1$). The dual of a line given by two points is thus simply the cross product.

We can geometrically verify that this is the same duality than defined before. Consider two points a (x,0) and b (x,1), which define a line parallel to the x-axis (Figure 285). The cross product gives the point with y= 1/a, as used above (Figure 286).

$$a \quad b \qquad a \times b$$
$$\begin{bmatrix} 1 \\ 0 \\ a \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ a \end{bmatrix} \qquad \begin{bmatrix} a \\ 0 \\ 1 \end{bmatrix}$$

4.4 Duality in Vector Space

The modules, and vector spaces so far, have been built upon a scalar multiplication where the scalar was the left and the vector the right argument:

```
(.) :: scalar -> vector -> vector.
```

These modules and vector spaces where left modules; the same construction is possible with a scalar multiplication, where the scalar is the right argument:

```
(.) :: vector -> scalar -> vector.
```

The vector space resulting from a right or left scalar multiplication are dual to each other.

In accordance with some of the literature (Mac Lane and Birkhoff 1991) we select a right module for the vector space. Points are expressed as '*column*' vectors, linear transformations are written before the point to which they apply $p' = T\,p$ (premultiplication), where p' and p are column vectors. Note that $T\,p$ looks like the application of a function T to p. For this choice, the lines are then row vectors, etc. Other texts on the subject use the other convention (row vectors for points,

postmultiplication for transformations) and some do not differentiate between points and lines and write for both row or column vector(Hartley and Zisserman 2000)).

4.5 FORMAL DEFINITION OF DUALTITY AS A DUOMORPHISM

A projective space can be defined as a tuple $T = (F, M, \vee, \wedge, V, U)$, where F is the set of all flats in the vector space s, M is the set of all projective maps (automorphism of s) an V, U are the units. The dual space $T^* = (F, M, \wedge, \vee, U, V)$ is isomorphic to *T(Stolfi 1991, 83)*. The isomorphism η from T to T^* must satisfy, for example

$\eta\ V = U, \eta\ U = V$
rank $(\eta\ a)$ = *corank a.*

5. REPRESENTATIONS OF POINTS AND LINES

Duality allows us to select between two representation for points and lines—the primal and the dual one.

5.1 POINTS AS VECTORS

Points are represented as homogenous column vectors (remember, the homogenous coordinate is first!).

$$p = \begin{bmatrix} 1 \\ p_x \\ p_y \end{bmatrix}$$

Figure 287: A point $p(p_x, p_y)$

5.2 LINES DEFINED AS LIST OF POINTS

Lines can be defined by enumeration of the column vectors of the two points that they are defined by. The dual of a line is a point, which is the row vector $v = p \times q$.

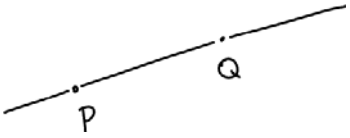


Figure 288: A line given by two points

$$p = \begin{bmatrix} p_0 \\ p_1 \\ p_2 \end{bmatrix} \quad q = \begin{bmatrix} q_0 \\ q_1 \\ q_2 \end{bmatrix}$$

$$l = \begin{bmatrix} p_0 & q_0 \\ p_1 & q_1 \\ p_2 & q_2 \end{bmatrix}$$

6. TRANSFORMATION OF A LINE IN DUAL SPACE

The transformation in dual space must correspond to the dual of the transformation in primal space. A point x on a line u before transformation must be on the line u' after transformation $x' = A\ x$; therefore $uT\ x = 0$ and $u'T\ x' = 0$. The line is transformed by the inverse transformation A^{-1}.

$$x' = Ax$$
$$u\,x = 0$$
$$u'x' = 0$$
$$u'Ax = 0$$
$$u\,A^{-1}A\,x = 0$$
$$u' = u\,A^{-1}$$

Note: if space and dual space are identified and points and lines represented by the same vector (as many texts do), then the transformation is the contragradient transformation, that is, transpose of the inverse transformations $(A^{-1T} u^T) = u'^T = (uA^{-1})^{T)}$.

7. LATTICES FOR GEOMETRIC OBJECTS

The usual program for geometric studies classifies objects and the applicable operations by dimension, that is, point, lines, and areas, and introduces with each additional dimension new objects and operations. Menger suggested a dimension independent program based on joining and intersecting generalized geometric objects(Blumenthal and Menger 1970, 135). It initiated investigation in an algebraic structure, which became generalized to include other similar structures and is called now Lattice theory(Birkhoff 1967).

A lattice is an algebraic system with two operations, called join and meet (sometimes written as sum for join (+) and product for meet (*)).These operations are commutative, associative—like a group—and absorptive. A lattice may have two distinct elements—called top $\top$ and bottom $\bot$ (sometimes written as 1 and 0, universe and vacuum) such that the regular axioms for unit elements are satisfied; if a lattice has these distinct elements, then they are unique.

$a \vee 0 = 0 \quad a \wedge 0 = a$
$a \vee 1 = 1 \quad a \wedge 1 = 1$

Lattices are dual with respect to join and meet and top and bottom.

With this definition of lattice, a dimension-independent description of the geometry of incidence can be achieved, but it is not sufficient to deal with orientation. The line from A to B is not differentiated from the line B to A, which makes it impossible to say that a point P is left of the line (Figure 289).

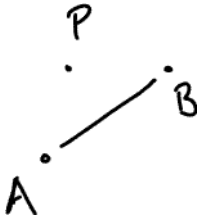
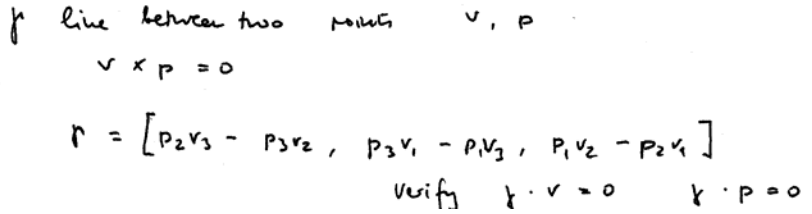


Figure 289Point P is left of A to B

Stolfi gives a description of a anti-commutative lattice-like theory, where $a \vee b \neq b \vee a$ and $a \wedge b \neq b \wedge a$, which can represent an oriented projective space. It is a generalization of Menger's approach with an orientation added. Lines have an orientation and there is an operation opposite to convert a line to the line with the opposite orientation. A description of the theory for the general n-dimensional case follows in the next chapter but meet and join in the following sections are understood as following the rules of this anti-commutative lattice.

7.1 JOIN OF POINTS GIVES LINE

A join of two points gives a line (see section 4 above), represented as two column vectors. The representation of the line as a (dual) point is the cross product of the two vectors that represent the two points. This can be checked, as the two points are on this line (i.e., the inner product of line and point is zero).



Join is only defined if the two points are distinct; if the same point appears twice, then the resulting is the homogenous triple (0, 0, 0), which does not represent any real point (this is different from lattice theory, where $a \vee a = a$). This observation can be used to determine if two points are the same:

a = b iff a x b = (0, 0, 0).

7.2 MEET GIVES INTERSECTION OF TWO LINES

The meet of two lines is the common part, i.e. the intersection point. The meet is the dual of the join, thus the $l \wedge n = l' \vee n'$, where l', n' are the duals of l and n. If the two lines are given by points a,b and c,d then the dual of the lines $l' = a \times b$, and $n' = c \times d$. Meet is expressed in terms of join and dual operations, which both translate to the cross product.

$p = l \wedge n$
$p' = l' \vee n'$
$p' = (a \times b) \vee (c \times d)$
$p = (a \times b) \times (c \times d)$

One can verify that p is on line l and m by checking

$p \cdot l = 0$ and $p \cdot m = 0$.

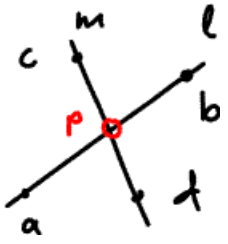


Figure 290: A point p = l ∩ m = l ∧ m

The duality between points and lines and the correspondence between join (connecting) and meet (intersecting), leads to commutative diagrams like Figure 291, which show that only one of the two operations and duality must be given to construct the other. Join of two points translates to cross product × and is therefore the preferred implemented operation.

$a \wedge b$ = dual ((dual a) $\vee$ (dual b))
$a \vee b$ = dual ((dual a) $\wedge$ (dual b))

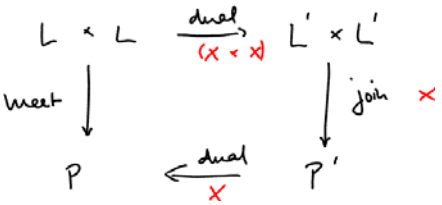


Figure 291: Duality

8. POINT—LINE RELATIONS:

Two relations between points and lines are derived from vectors. They are widely used and are related to the more general Matroid theory(Knuth 1992; Oxley 1992; Björner 1999):

- Which side of a line is a point?
- Distance of point from line?
- Is point inside of circumcircle of three points?

8.1 WHICH SIDE OF A LINE?

A method to determine if a point is left or right of a line (Figure 294) is to compute the determinant from the two points defining the line with the third point. The determinant gives twice the area and is signed: The determinant is positive if the three points are encountered in anticlockwise (positive) order and therefore the point c left of the line a to b. It is 0 if the three points are collinear.

$$ccw(a,b,c) \equiv \det\begin{bmatrix} 1 & 1 & 1 \\ x_a & x_b & x_c \\ y_a & y_b & y_c \end{bmatrix} > 0$$

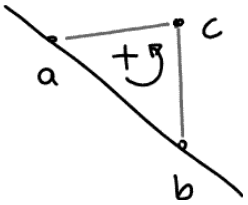


Figure 294: Point c is left of a – b

If the line l is given by its dual point, then we can use the triple product:

det (a,b,c) = triple (a,b,c) = triple (c, a, b) = c . (a x b) = c . l

8.2 DISTANCE OF POINT FROM LINE

The distance of a point from a line can be determined by division of the above determinant by the distance between the two points of the line.

8.3 INCIRCLE TEST

A test whether a point is inside a circle determined by three points will become important later (chapter 30xx). Given three points *ABC*, not collinear, incircle (*A,B,C,D*) is true, if *A B C* define in clockwise order a triangle and the point *D* is inside the circumcircle of this triangle. This is equivalent to test

Angle ABC + Angle CDA < angle BCD + angle DAB.

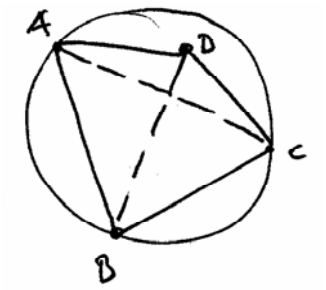


Figure 295: Incircle test

The test can be written as a determinant (for details see Guibas and Stolfi(Guibas and Stolfi 1987), where the sign of the determinant must be the same as for the ccw predicate(Knuth 1992):

$$\text{incircle}(A,D,C,D) = \begin{vmatrix} x_A & y_A & x_A^2+y_A^2 & 1 \\ x_B & y_B & x_B^2+y_B^2 & 1 \\ x_C & y_C & x_C^2+y_C^2 & 1 \\ x_D & y_D & x_D^2+y_D^2 & 1 \end{vmatrix} > 0$$

This determinant gives the same formula as when we compute the center of the circumscribing circle for the points *A, B, C* and then compute the distance from this center to one of the points and to the new point. The derivation is easier, if we use a local coordinate system with *A = (0,0)* and translate all other points to this system.

Center of circumscribing Circle $X(x/y)$

for Points $A\ (0/0)$

$B\ (x_B/y_B)$

$C\ (x_C/y_C)$

$\text{dist}^2 AX = \text{dist}^2 BX = \text{dist}^2 CX$

$(\text{dist}^2 AX)^2 = x^2 + y^2$

$(\text{dist}\ BX)^2 = (x_D - x)^2 + (y_D - y)^2 = x_D^2 - 2x_D x + x^2 + y_D^2 - 2y_D y + y^2$

$(\text{dist}\ CX)^2 = (x_C - x)^2 + (y_C - y)^2 = x_C^2 - 2x_C x + x^2 + y_C^2 - 2y_C y + y^2$

$+ (\text{dist}\ AX)^2 + (\text{dist}\ BX)^2 = 0 \qquad (\text{dist}\ AX)^2 - (\text{dist}\ CX)^2 = 0$

$2x_D x + 2y_D y - x_D^2 - y_D^2 = 0$

$2x_C x + 2y_C y - x_C^2 - y_C^2 = 0$

$$x = \frac{-y_D(x_C^2+y_C^2) + y_C(x_D^2+y_D^2)}{2(x_D y_C - x_C y_D)}$$

$$y = \frac{-x_D(x_C^2+y_C^2) + x_C(x_D^2+y_D^2)}{2(x_D y_C - x_C y_D)}$$

Substitute dist Ax < dist Px

$$\text{dist } Px - \text{dist } Ax > 0$$

$$(x_p - x)^2 + (y_p - y)^2 - x^2 - y^2 > 0$$

$$x_p^2 - 2x_p x + x^2 + y_p^2 - 2y_p y + y^2 > 0$$

$$2(x_B y_C - x_C y_B)(x_p^2 + y_p^2) + 2x_p\, y_B (x_C^2 + y_C^2) - 2x_p\, y_C (x_B^2 + y_B^2)$$

$$- 2y_p\, x_B (x_C^2 + y_C^2) + 2y_p\, x_C (x_B^2 + y_B^2) > 0$$

$$x_B y_C (x_p^2 + y_p^2) - x_C y_B (x_p^2 + y_p^2) + x_p y_B (x_C^2 + y_C^2) - x_p y_C (x_B^2 + y_B^2)$$

$$- y_p x_B (x_C^2 + y_C^2) + y_p x_C (x_B^2 + y_B^2) > 0$$

9. CONCLUSIONS

Coding intersection of lines is one of the more tricky parts of geometric processing(Goldenhuber 1997). The solutions achieved through the use of projective geometry are clean and elegant, when compared the solutions which need complicated tests. The principles found here can be generalized to *n*-dimensional space. The arguments here for geometric objects in the *2d* plane and the corresponding *3d* projective space use the cross product for the transformation to the dual. Cross product is defined for *3d* vectors only and the next chapter will need a generalization to *n*-dimensions.

REVIEW QUESTIONS

- What is duality? Explain with already known algebras (set theory).
- What is the meaning of homogenous?
- What is a lattice structure? In what sense is it dual?
- How do the operations meet and join apply to geometry?
- Explain the duality of points and line?
- Why are we using projective space?
- Give the formulae for the intersection of two lines given by points.
- What is the difference between the geometric program by Hilbert compared to the approach by Menger?

Chapter 20 GENERALIZATION TO N-DIMENSIONS: FLATS

In the previous chapter we have seen how projective geometry and duality leads to a simple formula for the calculation of the intersection. This chapter will generalize the solution found for lines in *2d* space in the previous chapter to *n*-dimensional infinite geometric objects. We use Menger's approach to investigate joins and meets of subspaces. These subspaces, considered as geometric objects, will be called flats. In preparation for geometric operations defined later, the space investigated is oriented and the algebra is lattice-like, specifically anti-commutative. The chapter concludes with a single formula for all intersection calculations, whatever the dimension of the space and the geometric element, using dual and join as the fundamental building blocks.

The solution found for *2d* objects uses extensively the cross product, which is defined only for *3*-vectors. This restricts the formulae given to *2d* (*3d* homogenous) geometry. This chapter starts with the identification of vector and matrix operations that are dimension independent and generalizes vector (cross) and triple product from *3* to *n*-dimensions. This leads to a generalized cross product *gpc* on *nearly square matrices* (*n* by *n-1*). With this operation, the dual of *k*-dimensional objects in *n*-dimensional space can be defined for all $k < n$ and operations to compute the intersection follow immediately.

1. SUBSPACES OF *N*-DIMENSIONAL SPACE

In the following we will assume an *n*-dimensional, oriented space ($n >= 2$). In *2d* we had infinite geometric objects point and line, which were dual to each other. The approach used in section 19.2xx defining points in a *2d* space as the lines through the origin of a *3d* space (O-lines) can be generalized. To represent a n-dimensional projective space *P*, we use a *n+1*-dimensional vector space *V* and a mapping *P = P(V)*. The subspaces of this *n+1*-dimensional vector space each contain the origin; otherwise they would not be the sub-spaces. We can visualize them as the O-lines, O-planes etc. of Menger's model.

There is a morphism P from the sub-spaces of the vector space V: P = P(V)

A subspace of *m*-dimension is defined by *m+1* point; a line (*1*-space) is defined by *2* points, a plane is defined by *3* points

etc. *k+1* points in an *n*-dimensional projective space *V* (with *k* <= *n*) define a *k*-dimensional vector subspace of *V* of *k+1* dimension. We will call a subspace of *V* a *flat*, specifically *k-flat* where *k* is the dimension of the subspace (and thus the rank of the projective geometric object). A *k* -flat has dimension *k* and codimension *n-k*. Subspaces with dimension *n-1*, i.e., with codimension *1*, are called *hyperplanes*; they are dual to points.

A k-flat *is a* k-*dimensional subspace of the projective space.*
A hyperplane is the dual to a point.

Subspaces are partially ordered by an inclusion relation: $S \subseteq T$, that is, *S* is a vector subspace of another vector subspace *T*. With this inclusion relation, the subspaces form a lattice, where the meet is the intersection and the join is the direct sum of the two spaces (Mac Lane and Birkhoff 1991, 594/5). The mapping to from the *k+1*-dimensional vector space to the corresponding *k*-dimensional projective spaces *T -> P(T)* preserves this inclusion and is an isomorphism of lattices.

2. REPRESENTATIONS FOR LINES AND PLANES IN 3-SPACE

For comparison and reference, I include here a description of the often used representations of lines and planes in 3-dimensional space.

2.1 REPRESENTATION OF LINES IN 3D SPACE

A line can be defined as the geometric locus of all points collinear with given two points (Figure 296). Using the fact that the vector product of two collinear vectors is 0 gives:

(b-a) x (p-a) = 0, or (p-a) x u = 0 where u = b – a.

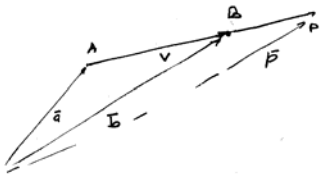


Figure 296: A line in 3d space

2.2 REPRESENTATION OF PLANES IN 3D SPACE

In addition to the description as a function

*z = f (x, y) = a * x + b * y + c,*

which cannot represent vertical planes, a plane is defined as all points *p* for which the following equation in two parameters is valid.

$p = a + \lambda \, . \, v + \mu \, . \, w = a + \lambda \, . \, (c\text{-}a) + \mu \, . \, (b - a)$

where *v* and *w* are two vectors in the plane. Using that the triple product for coplanar vectors is 0, follows

triple product (v,w,p) = 0.

The definition using an orthogonal vector $n = v \times w$ (Figure 297) gives $n \, . \, p = 0$.

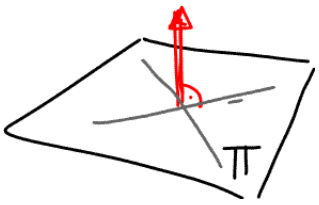


Figure 297: A plane defined by the normal on it

3. JOIN AND MEET: DIMENSION INDEPENDENT GEOMETRIC OPERATIONS

Only few geometric operations are possible in spaces of arbitrary dimension. The workhorse so far was cross product, which exists only in *3d* space. Independent of dimension are:

- join: the construction of linear subspaces of higher dimension from points or generally from other flats, and
- meet: the intersection of two flats that results in another flat.

3.1 JOIN: CONSTRUCTING FLATS FROM POINTS OR OTHER FLATS

Geometric object of higher dimension are constructed from objects of lower dimension: A line is constructed with two points, a plane with three points (or a point and a line). *k* points in *n* space (in general position) form a *k*-dimensional subspace, a *k-flat*. This join operation, which combines two flats to produce a flat of higher dimension has the usual properties. The units are the flat, defined with zero points (dimension -1), called *vacuum* and the (*n+1*) flat, which is the *n*-dimensional space, called *universe*.

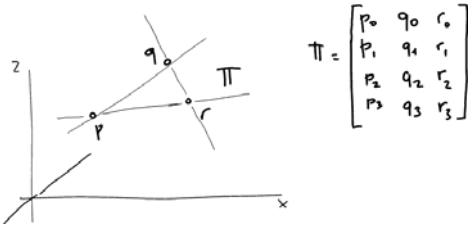


Figure 298: Construction of geometric objects from points

Join is only defined, if the two objects have no common part. Geometrically evident is the rank of the result of the join:

$$\dim(a \vee b) = \dim a + \dim b \quad \text{if } a \wedge b = 0$$

The primary representation for projective *k-flats* in *n* space is the matrix with *k* columns and *n+1* rows, which results from joining the column vectors standing for the points into a matrix (Figure 298).

3.2 MEET: INTERSECTING TWO FLATS

Intersection of two flats gives another flat. Meet is dual to join. The direct calculation of the is avoided here and replaced by duality:

dual (a meet b) = (dual a) join (dual b)

and obtain

a meet b = dual ((dual a) join (dual b)).

4. DUALITY IN *N*-DIMENSIONS

Duality must be self-inverse: *dual* . *dual* = *id*. It is useful to introduce rank and co-rank to explain what objects can be dual to each other.

4.1 Rank and Co-rank

The rank of an object is the number of points used for its definition, which is its dimension plus one. A *k*-flat has therefore rank *k+1* (and dimension *k*). The co-rank is the same as the co-dimension and counts how many points must be added to get the universe.

rank = 1 + dim
rank + corank = (1 + dim) + (n – dim) = n + 1

Ranks:
Point = 1-flat
Line = 2-flat
Plane = 3-flat

4.2 Dual object

The rank of the dual of an object is its co-rank and the co-rank of the dual of an object is its rank: *rank . dual = corank; corank . dual = rank*. This duality applies to *2*-space and gives the previously encountered duality between points (rank *1*, corank *2*) and lines (rank *2,* corank *1*).

In 3d space, points (rank 1) are dual to planes (rank 3, corank 1). Lines are dual to lines (rank 2, corank 2).

4.3 Duality of join and meet

The rank and corank of the result of a *join* or a *meet* are dual if *a* and *b* are disjoint:

rank (a ∨ b) = rank (a) + rank (b)
corank (a ∧ b) = corank (a) + corank (b)

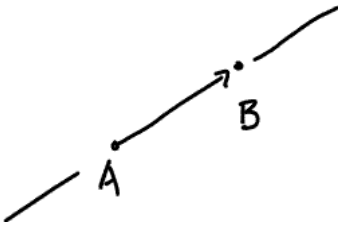


Figure 299 Line A to B is oriented

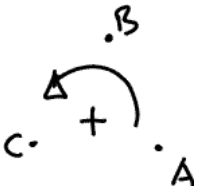


Figure 300: Plane given by A, B, C is oriented positively

5. Orientation

Each flat has an orientation. For lines, the orientation is the direction of the line from first to second point (Figure 299); for planes, the orientation is the sense of turning given by the order of the three points (Figure 300).

The operation reverse ¬ converts a flat in the corresponding flat with the other orientation.

(line a b) = ¬ (line b a)
(plane a b c) = ¬ (plane a c b)

The flats occurring in a GIS are orientable, but the projective plane, into which they are mapped, cannot be consistently oriented. It behaves somewhat like a Moebius strip (Figure 301), which is the prototypical non-orientable surface. If geometry is restricted to the non-ideal parts of the projective plane, then orientation can be consistent (for more details see(Stolfi 1991). This means that we cannot construct figures that include the line at infinity and must not transform figures through this line.

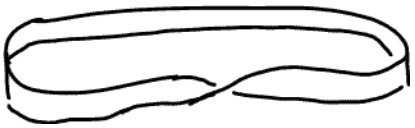


Figure 301: A Moebius strip - has only one edge and one surface!

6. THE ANTI-COMMUTATIVE LATTICE OF ORIENTED FLATS

Lattices are commutative ($a \wedge b = b \wedge a$; $a \vee b = b \vee a$), but the join operation used here is anticommutative to respect the orientation.

Anti-commutative Lattice $<L, \wedge, \vee, V, U, \neg>$

anti-commutative	$p \wedge q = \neg(q \wedge p) = \neg q \wedge p$ *(for p, q of rank 1)*	$p \vee q = \neg(q \vee p) = \neg q \vee p$
associative	$r \wedge (s \wedge t) = (r \wedge s) \wedge t = r \wedge s \wedge t$	$r \wedge (s \wedge t) = (r \wedge s) \wedge t = r \wedge s \wedge t$
units	$V \vee a = a = a \vee V$	$V \wedge a = V = a \wedge V$
	$U \vee a = U = a \vee U$	$U \wedge a = a = a \wedge V$

Join and meet are only defined when the operands are disjoint; Stolfi suggests that implementations make the operations total and introduce a "undefined" object 0 (respective one for each rank). This can be used to test if two flats are disjoint by computing the join and then check whether the result is 0(Stolfi 1991 45).

The rank of a join of two disjoint flats is the sum of the ranks of the flats; the co-rank of the meet of two flats is the sum of the co-ranks, but we can also say that meet lowers the rank of the first flat by the co-rank of the second flat:

$corank\ (a \wedge b) = corank\ (a) + corank\ (b)$
$rank\ (a \wedge b) = rank\ (a) - corank\ (b) = rank\ (b) - corank\ (a)$

For p and q flats with rank a and b and co-rank a' and b' the commutative law is

$p \vee q = \neg^{(a+b)}(q \vee p);$
$p \wedge q = \neg^{(a'+b')}(q \wedge p);$

this means each swap of two of the defining points changes the orientation (reminds of the change in sign of matrix determinants and swap of columns or rows).

7. THE DUAL OF FLATS

The dual of a line (a *1*-flat) in *2d* space is defined as the vector such that all points p of the line l given by two vectors a and b are given by the equation $l' \,.\, p = 0$ where $l' = a \times b$ is the dual of the line . How to generalize this to *n-1* flats in *n* space? This means first, how to replace the cross product with an operation that is available in all dimensions? We will approach this problem in four steps, (1) determining the dual of a hyperplane

(*n*-flat) gives point in *n*-space, (2) the dual of a line in *3*-space, (3) the dual of a line in *n*-space and (4) a fully general solution for *k*-flats in *n*-space.

7.1 DUAL OF HYPERPLANE GIVES POINT

By definition of hyperplanes they are dual to points. In *n*-dimensional projective space, a hyperplane has *n-1* dimension (it is an *n*-flat), rank *n* and co-rank *1*. Its dual has rank *1* and is a point.

We map the projective *n* space to subspaces of *n+1* vector space (homogoneous coordinates). The hyperplane is given as a join of *n* vectors, each with *n+1* coordinates. From the equation for coplanarity in *n* space, we find that for any point *p* in this hyperplane the determinant of the join of *h* with *p* is *0*. This formula is independent of dimension. Can we use it to determine *p*?

$$det\ (h \vee p) = 0 = det\ |x_1, x_2, x_3 \ldots x_n, p| = 0$$

Nearly square matrix:
A matrix with one row more than columns, respectively one column more than rows.

The hyperplane *h* is defined by *n* points; when joined, this gives nearly a square matrix *n by n+1*, resulting from the join of *n* point vectors in homogenous coordinates. The expansion of the determinant $det\ |x_1, x_2, x_3 \ldots x_k, p|$ for the last column gives a vector *h'*, for which $h' \cdot p = 0$ (see before cofactors of a matrix; chapter 10).

$$h = [p_1, p_2, p_3 \cdots p_n]$$

$$h' = gcp\ h = gcp\ [p_1, p_2, p_3 \cdots p_n]$$

The operation *gcp* (for generalized cross product) takes a *nearly square matrix* of *n (n+1)*-vectors $x_1, .. x_n$ and computes a vector $h' = gcp\ (x_1 \ldots x_k)$, such that $h' \cdot p = 0$. It is computed as the values of the subdeterminants of the nearly square matrix, crossing out one row after the other. The computation of cofactors and the inner product is independent of dimension. Note that *gcp* has no inverse; it can only transform a join of points to a dual space, not the inverse.

$$h = [x_1, x_2, \ldots x_n]$$

$$det\ (h \vee p) = det\ [x_1, x_2 \ldots, x_n, p]$$

$$= det\ M$$

$$= M_1 \cdot p_1 - M_2 \cdot p_2 + \ldots M_{n+1} \cdot p_{n+1}$$

where M_i is determinant of M with i^{th} row crossed out

$$gcp\ (x_1..x_n) \cdot p = det\ (p, x_1 \ldots x_n)$$

Primal to dual transformation = gcp

$$gcp\ h = gcp\ [x_1, x_2, \dots x_n]$$
$$= [\det h_{\bar{1}}, \det h_{\bar{2}}, \dots \det h_{\overline{n+1}}]$$
where $h_{\bar{i}} = h$ with row i crossed out

Footnote:
The computation is connected to the Hodge operator and the Eddington or Levi-Civita tensor(Faugeras 1993, 160-62); it follows from the Kronecker or outer product when computing the products of the base vectors such that any product of $e_{(n-1)}$ factors, where any of the e_i appears twice is 0 and we identify the product $e_1 \dots e_{j-1}, e_{j+1}, .. e_n = e_j$. This gives for the regular cross product the products of the base vectors as $e_1 * e_2 = e_3$, etc. The argument using the expansion of the determinant for the missing first column and the analogy to the triple product seems simpler and suggest the generalization following in the next section.
endFootnote

This derivation gives for to the *2d* projective space the cross product:

$$l = p \vee q$$
$$l' = gcp\ [p, q]$$
$$= gcp \begin{bmatrix} p_1 & q_1 \\ p_2 & q_2 \\ p_3 & q_3 \end{bmatrix}$$
$$= \left[\det \begin{bmatrix} p_2 & q_2 \\ p_3 & q_3 \end{bmatrix}, -\det \begin{bmatrix} p_1 & q_1 \\ p_3 & q_3 \end{bmatrix}, \det \begin{bmatrix} p_1 & q_1 \\ p_2 & q_2 \end{bmatrix}\right]$$
$$= p \times q$$

A hyperplane *h* of dimension *n-1* can be represented in homogenous coordinates by a *n by n+1* nearly square matrix. It includes all points *p*, such that *det (h ∨ p) = 0* (observe that *(h ∨ p)* is square). It has a dual *h' = gcp h*, which is represented as an *n (row)* vector, such that *h' . p = 0*.

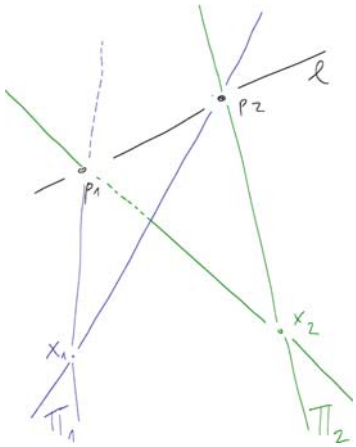


Figure 302: Construction to compute the dual of a line in 3d

Figure 303: Dual of the line, connecting the dual points of the two planes Π_1 and Π_2

7.2 DUAL OF LINE IN 3-SPACE

Lines in *3d* projective space have codimension *2*, they are *2*-flats. The dual of a line is again a line, because a line has rank *2* and corank = codimension *2*.

The determine the dual of a line in 3d projective space, consider the situation visualized in (Figure 302). One could get the dual of the line p1, p2 by determination of the dual p1' of p1 and p2' of p2. Then the intersection of these two planes p1' and

p2' is the dual l' of l. Unfortuantely, this requires the computation first of the dual plane to a point remember gcp does compute only the dual point to a plane) and of an intersection (meet), which we want to avoid and replace by the easy to compute join.

A second and successful approach to determine the dual of a line *l* given by two points p_1, p_2 is obtained by considering two planes Π_1 and Π_2 through this line and two arbitrary points x_1 and x_2 (different from p_1 and p_2). The dual points for these two planes are $\underline{\Pi_1} = gcp\ (l \vee x_1)$ and $\underline{\Pi_2} = gcp\ (l \vee x_2)$. The dual of the line is the join of these two points (Figure 303).

$$l' = \Pi_1 \vee \Pi_2 = (gcp\ (l \vee x_1)) \vee (gcp\ (l \vee x_1)) = [gcp\ [p_1, p_2, x_1], gcp\ [p_1, p_2, x_2]]$$

The approach results in a description of the line with *2 * 4* parameters, when only 4 are necessary. The result is one out of an equivalent class of descriptions of the same dual line and is influenced by the choice of the arbitrary points x1 and x2, for which is required that $det\ |x_1\ \Pi_1| \neq 0$ and $det\ |x_2\ \Pi_2| \neq 0$.

Footnote: The method proposed by Plücker would use only 6 Plücker coordinates. The approach presented here and the methods using Plücker coordinates produce both results that are just one element of an equivalence class. The major difference in the approach presented here is that the arbitrary elements are introduced initially when selecting the two points x_1, x_2. Plücker gives a general solution that is then constrained with the so-called Plücker constraint. The approach suggested here is similar to the approach selected in (Leonardis and Bischof 1996) for a different problem; it is attractive, because the x_1, x_2 can be selected to produce numerically good conditions for the resulting matrices.

Endfootnote

7.3 DUAL OF LINE IN N-SPACE

The generalization of the approach discussed for *3d* projective space to *n*-dimensional projective space is immediate. A line in *n*-dimensional space is defined by *2* points (*n+1* homogenous coordinates), it has rank *2*. The dual has *co-rank 2 = rank (n+1-2) = rank (n-1)*. For example, a line in *4d* has as a dual a plane with *rank 3 = 4-1*. To determine the dual we have therefore to identify *n-1* dual points.

To determine a dual point with *gcp*, we have to join the two points *p1, p2* of the line with *n-2* arbitrary points $x_1 \ldots n_j$ and

compute the dual point $q = gcp\ (p_1, p_2, x_1 \ldots x_j)$. For example, for a line in *4d*, this requires *2* arbitrary points x_i. This must be repeated to obtain the *n-1* points which joined together give the dual flat.

7.4 DUAL OF K-DIMENSIONAL OBJECTS IN N-DIMENSIONAL SPACE

In general, the dual for a *k-flat* in *n*-dimensional space is a *(n+1-k)*-flat and is determined by *(n+1-k)* dual points. For each such point, the *k* vectors of the flat must be joined with *n-k* (linearly independent) vectors to form the nearly square *(n* by *n+1)* matrix of which the generalized cross product *gap* is computed to obtain one of the dual points which determine the dual flat. Joining these dual points give the dual flat.

Footnote:

The generalization of the above approach to higher dimension is attractive, because the number of Plücker coordinates in spaces of higher dimension grows rapidly for 4-dimensional space (i.e., homogenous coordinates of dimension 5). Stolfi suggests a mixed representation(1991, 195/196), which represents *k-flats* in *n* space for $k < n/2$ as joins of the points used for the construction and for $k > n/2$ their duals (which use *(n-k)* points). Stolfi proposes a more compact 'reduced simplex representation'. To achieve this, a direct calculation of intersection would be necessary for cases where the primal representation is more compact than the dual representation, which is the case for all flats k <= n/2; the intersection is then computed, for example, using s Single Value Decomposition (SVD) from which the nullspace of the two intersecting subspaces results (Hartley and Zisserman 2003 70)

End footnote.

7.5 JOIN AND MEET FOR FLATS

With the operation to determine the dual for any flat independent of the dimension of the flat or the space and the join operation given by simple collection of column vectors of points into a matrix we have the key to a generalized intersection operation which gives the dual of the intersection:

$$(a \wedge b)' = (a' \vee b')$$

7.6 CONSIDERATION OF TYPES

We have selected column vectors to represent points and row vectors for the representation of lines. Dual points are lines, therefore dual points are also row vectors and duality transposes rows to column vectors and in general a k-flat, which is a k by n+1 matrix into a n+1 by (n-k) matrix (k<= n).

Join of dual elements combines them vertically and the *gcp* of a dual geometric object is the transposed *gcp* of the transposed object : $gcp\ \underline{r} = (gcp\ \underline{r}^T)^T$.

$\bar{p}, \bar{q}$: dual points

$\bar{p} = [p_1, p_2, \ldots p_n]$

$\bar{q} = [q_1, q_2, \ldots q_n]$

$$\bar{p} \vee \bar{q} = \begin{bmatrix} p_1, p_2, \ldots p_n \\ q_1, q_2, \ldots q_n \end{bmatrix}$$

Mathematicians and engineers tend to ignore the homomorphism, which embeds one type into another more complex one. For example it seems that vectors representing points and the dual of a line could be equated and we are tempted to write $a \,.\, b = a^T b$. Checking the types reveals the difference: the inner product yields a real number, whereas the result of the matrix multiplication is a matrix with a single element. It would be correct to write $a \,.\, b = det\ (a^T b)$. It is often useful to differentiate between row and column vectors, between scalars and matrices which have just a single element etc.

8. METRIC RELATIONS

Distance between two points, the volume (respective area) of the convex hull of a set of points, and the incircle relation are the generally useful geometric relations. They identify discrete relations to metric properties and are used to deduce the discrete relations from the continuous metric.

8.1 DISTANCE

The distance between two points is a property of a metric space. In a vector space, it calculated as the length of the vector between the two points. From the definition of distance follows that two points are equal if they have distance = 0. This translates in a test for equality:

dist (a,b) = sqrt ((a-b) . (a-b))
a == b = dist (a,b) ==0

8.2 VOLUME

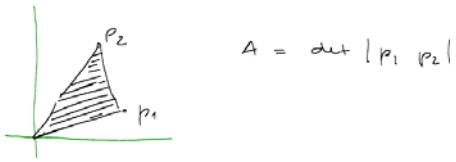


Figure 304: det is area spanned by vectors

The determinant of *n* vectors in *n*-space gives the area of the volume of the convex hull of these *n* linearly independent points and the origin (Figure 304). The area or volume of an *n+1* polytope given by *n+1* vectors in *n*-space is computed with the determinant in the homogeneous space (*n+1* vectors, with *h=1*).

If *n+1* points are given to describe a polytope in n-dimensional projective space, then the determinant is computed from their homogeneous (*n+1*) coordinates, divided by the product of the w_i. The polytope in the *n+1*-dimensional vector space has height *1* and therefore the value for the volume and the area is the same $v = h * a$ (Figure 305). A proof for the general case follows from the definition of the vector operations and homogeneous coordinates by simple algebraic computation.

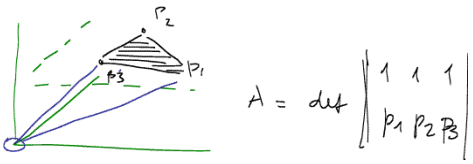
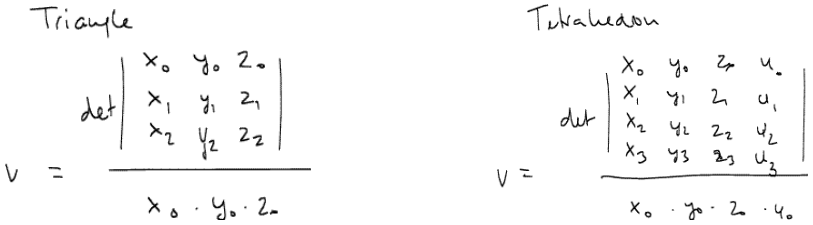


Figure 305: The area from 3 points



The volume computed with this formula is signed and has a positive value if the points are listed in counter clockwise order. The determinant can be used to test for the counterclockwise ordering of the points or to determine if a point is left or right of a flat. This is a generalization of the CCW predicate from chapter 19. The determinant is 0 for points that are collinear, resp. coplanar.

9. CONCLUSION

The introduction of a *generalized cross product* for *nearly square matrices*, together with *join* interpreted as building flats from points and *meet* as intersections gives a compact and dimension independent formalization of intersection of flats of any dimension. The solution avoids the use of Plücker coordinates and gives a uniform representation for all infinite objects, albeit not minimal, but the loss of storage space is today not a primary concern.

REVIEW QUESTIONS

- Three planes intersect in a single point. How can this be used for a test of *gcp* in *3d* space?

PART SEVEN BOUNDED GEOMETRIC OBJECTS

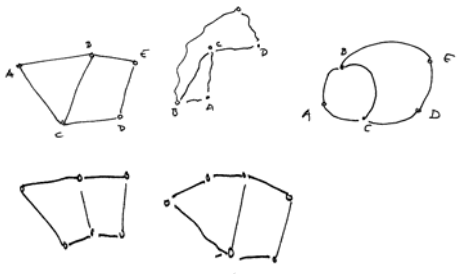


Figure 306: Five topologically equivalent figures

The previous chapter introduced points and flats, subspaces of limited dimension but infinite size and the geometry of intersections. In this part, finite objects, objects with boundaries are introduced; finite, bounded objects consist of an inside and outside, which 'hang together'. This part explores this 'hanging together' as topological property, considering neighborhoods and transformations that transform a neighborhood into a neighborhood (Figure 306). These topological transformations are a much larger class of transformations than the general linear transformations (discussed in chapter 10), but including these.

The treatment follows the dimension independent approach to geometry. As far as practical, the discussion is not in terms of objects of a specific dimension, but stresses operations and relations that can be explained independent of the dimension of the objects or the space in which they are included.

The first chapter in this section introduces topology: space as sets of points and their boundaries in continuous space. The second chapter discusses topological relations. The following part combines then the topological concepts with algebraic approaches.

Chapter 21 POINT SET TOPOLOGY

In this chapter space is conceptualized as an infinite set of points. Space is a base category of human experience and perception. Everybody experiences space and time as continuous; this chapter should capture this experience as a formal property of point sets. Continuity does play a role in the axiom system of Euclid: it is postulated that there is a middle point between two points; this is essentially stating that the line is continuous. This chapter generalizes this notion of continuity from the continuity of a line to the continuity of space in general.

Topology captures the notion of "continuous" independent of numbers. There are no breaks in space—only breaks in the geometric structure we impose on it and there are no breaks in time—only the structure we impose on it creates distinct objects. This is an application of Jordan's curve theorem that separates objects with a boundary from its environment (see section xx).

Topology is used in a GIS in many ways: topology defines the relations between an area and its boundary or the relation between two areas. We may have a list of countries belonging to the EU—can we determine the boundary of the EU? What countries are neighbors? Topology deals with invariant properties of figures. All the geometries in Figure 306 are equivalent. Each figure can be transformed to any of the other ones by a homeomorphic transformation

Point set topology consider space as a collection of points. There are topologies for spaces represented with a finite number of points, so called discrete spaces; they could be potentially very interesting for GIS but have not yet been explored (see especially studies of lattices, which are also used in crystallography). This chapter concentrates on space formed by an infinite number of points. The next part will show the combination of algebraic methods with topology to achieve finite representations of spatial objects.

homoemorphism =
topological transformation

A homoemorphism is continuous transformation and has a continuous inverse

ɔrphism)

Figure 307: The same topology on a balloon when inflated and deflated

Topology = geometry on a balloon

1. TOPOLOGY IS BRANCH OF GEOMETRY

Topological relations are the relations that remain the same, remain invariant, under continuous transformations. Topology is the branch of mathematics, which discusses topological relations. 'Topology is geometry on a balloon' is a popular expression of what is studied here (Figure 307). Point set topology is based on the notions of neighborhood and homoemorphic transformations, which are continuous transformations which have an inverse which is also a continuous transformation.

Set theory with the major operations union and intersection has been introduced before (chapter 5 xx). Most sets encountered before had a finite number of elements in them; the sets used to represent continuous space are typically sets with an infinite number of points in them. Only an infinite number of dimensionless points together capture our perception and experience with continuous space. The impossibility to directly represent this infinite number of points in a computer is the source of a large part of the difficulties in implementing GIS.

2. DEFINITION OF NEIGHBORHOOD AND CONTINUOUS TRANSFORMATION

Topology is the geometry that investigates properties that remain invariant under topological transformations, that is, transformations that preserve neighborhoods. An axiomatization of topology starts with capturing the properties of a neighborhood and then leads to the definition of continuous transformations as transformations that map neighborhoods into neighborhoods. Neighborhoods are fundamental for the definition of open and closed sets and other topological concepts. Alternatively, one can select open sets as fundamental and then define neighborhoods from them.

Figure 308: A 1-, 2- and 3-sphere

A neighborhood in usual *n*-dimensional space is the homoemorph image (topologically equivalent image) of an *n*-sphere; a *3*-sphere is a ball, a *2*-sphere is a disk, a *1*-sphere is an interval (Figure 308).

2.1 AXIOMS FOR NEIGHBORHOODS

Topology introduces the concept neighborhood as a fundamental concept and defines it with the following axioms(Alexandroff 1961, 9):

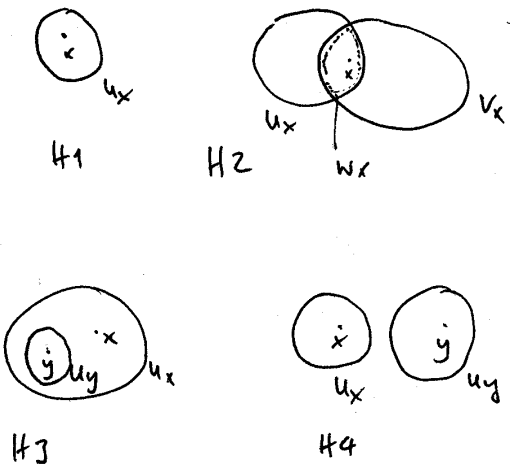


Figure 309: Illustrations for the axioms for a neighborhood

Neighborhood Axioms

H1: To each point x there corresponds at least one neighborhood U(x);
each neighborhood U(x) contains the point x.

H2: If U(x) and V(x) are two neighborhoods of the same point x,
then there exists a neighborhood W(x), which is a subset of both.

H3: If the point y lies in U(x), there exists a neighborhood U(y), which is a subset of U(x).

A space with neighborhoods is a topological space.

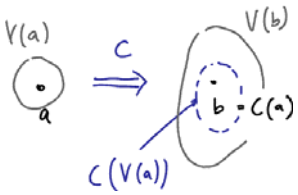


Figure 310: Topological transformation

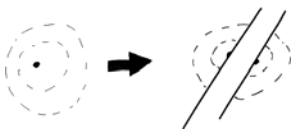


Figure 311: Not a topological transformation

2.2 DEFINITION OF CONTINUOUS TRANSFORMATIONS PRESERVING NEIGHBORHOODS

Transformations that do not change the neighborhood are called continuous: Neighborhoods are mapped to neighborhoods. The exact definition is:

"A mapping *f* of a topological space *R* onto a (proper or improper) subset of a topological space *Y* is called continuous at the point *x*, if for every neighborhood *U(y)* of the point *y= f(x)* one can find a neighborhood *U(x)* of *x* such that all point of *U(x)* are mapped into points of *U(y)* by means of f. If *f* is continuous at every point $f \in R$, it is called continuous in *R*." (Alexandroff 1961, 9)

If there is a neighborhood around a point—whatever small—which is not preserved (i.e., is not mapped to be contained in a neighborhood around the mapped point) then this points is a *discontinuity* point.

2.3 DIMENSION OF A SPACE

The dimension of a (usual) space is defined as the dimension of the neighborhoods. Neighborhoods are homoemorph for example, to a *2d* disk or a *3d* sphere.

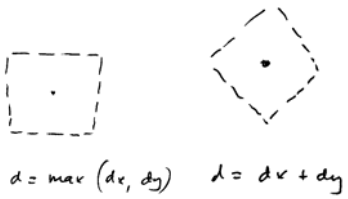


Figure 312: Two different distance functions give different neighborhoods(but the same topology)

Topological transformation preserve dimension.

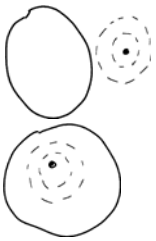


Figure 313: Interior point

Figure 314: Exterior point

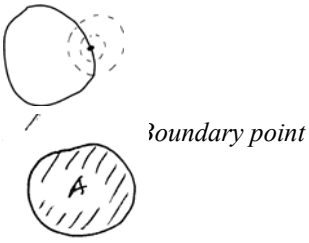


Figure 316: A is closed

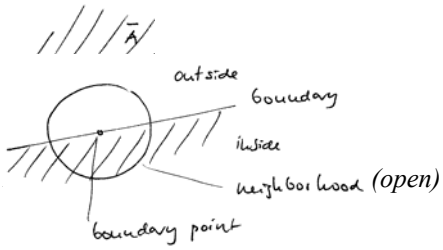


Figure 318: Outside, boundary, and interior

Boundary—All points that are neither interior nor exterior.

3. METRIC SPACES

A space in which there is a distance defined for any pair of points is called a metric space. A definition of a distance function induces a concept of neighborhood to a space. Different distance functions (see chapter xx) give different neighborhoods but not necessarily different topological spaces (Figure 312).

The natural topology for geographic space is the topology following from the ordinary Euclidean metric. Other topologies are possible, but seldom used in applications.

4. INTERIOR, EXTERIOR, AND BOUNDARY POINTS

It is useful to differentiate points inside, on the boundary or outside of a set:

- Interior Point (Figure 313): a point for which any sufficiently small neighborhood contains only points which are in the set.
- Exterior Point (Figure 314): a point for which any sufficiently small neighborhood contains only points which are not in the set.
- Boundary points (Figure 315) are those points, for which any neighborhood contains points that are inside and points that are not inside the set.

Definition boundary point: In any neighborhood of a boundary point—whatever small—there are points that are outside and points that are inside

5. BOUNDARY, INTERIOR, EXTERIOR

The notions of importance for practical work are *boundary*, *interior*, and *exterior*. The interior of a figure is the set of all interior points; the boundary of a figure is the set of all boundary points. The figure is the union of interior and boundary, and closed. The exterior is the complement of the figure.

Egenhofer has shown, how ordinary geometric relations like touch, intersect, etc. can be defined using only the notions of interior, boundary, and exterior (Egenhofer 1989) (see next chapter). For every geometric figure, operations to determine interior, boundary and exteriors will be required and then the determination of the topological relations as defined by Egenhofer follows from intersections of these parts of the figures.

6. OPEN AND CLOSED SETS

A set is called open, if it contains only interior points. A close set contains the interior points together with the boundary points. A set is close or open or neither of the two.

Open set = interior only
Close set = interior and boundary

Open and closed sets can be taken as fundamental for topology and a set of axioms given (S1 to S4). This is an alternative approach to the axiomatic foundation given before. The axioms H1 to H3 (above) follow from the axioms for open and closed sets; if H1 to H3 are assumed as axioms then S1 to S4 follow as theorems: the two theories are equivalent.

Open and Closed Sets (subset of M)

S1. The empty set and the set M are open

S2. The intersection of two open sets is open (as is the intersection of finitely many open sets)

S3. The union of open sets is open (finite or infinite union).

S4. A subset of M is a neighborhood of x if there is an open set O such that x is element of O and O a subset of M.

An open set is a set that does not include its boundary.

For subsets of a set the operation complement applies also to open and closed sets, such that the complement of an open set is closed and vice versa.

Complement

open a => closed (complement (a))

Closed set could be defined using the complement operation. The complement of open set is closed; a set is closed, if the complement is open. Sets can be half-open – being open in some place and closed in others.

Figure 319: Set A is half-open, A union B is closed

7. CLOSURE

The operation closure adds the boundary to a set, and converts it to a closed set. An already closed set is not changed, the operation is idempotent (*closure . closure = closure*). The closure of a half-open or open set is a closed set containing the given set. The axioms for closure give yet another axiomatic base for topological space::

---

topological space is a pair (X, closure), consisting of a set X and a mapping

closure : Powerset X -> Powerset X, such that for all a, b < X:

closure 0 = 0

a ⊆ closure a,

closure (closure a) = a

closure (a ∪ b) = closure a ∪ closure b

closed (closure a) = True
closure . closure = closure

8. CONNECTED

A space is connected if it cannot be written as a disjoint union of open sets; for most applications the simpler notion of 'path connected' is used: A figure is connected if between any two points is a path that is completely in the interior (Figure 320). Figure 321 gives an example of a space which is connected, but not path connected: there are no interior points in the line connecting the two parts, therefore no path 'in the interior' goes from left to right. Continuous mappings of connected spaces are connected; the union of non-disjoint connected spaces is again connected (Figure 322).

Figure 320: Connected figure

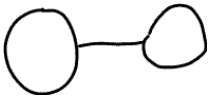


Figure 321 Connected but not path connected figure

Figure 322: The non-disjoint union of two connected spaces is connected

A region is simply connected, if it has no holes. Every closed path in a connected figure can be transformed homoemorphically to every other closed path; this is not possible in a figure, which is not simply connected (Figure 325).

Figure 323: Simply connected figure

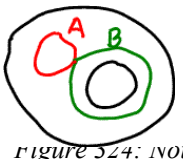


Figure 324: Not simply connected
Figure 325: Path A cannot be transformed to path B

Holes are crucial for the way objects can be used: Rooms are holes in buildings, as are windows, a needle without a hole is useless etc.(Casati and Varzi 1994). Lakes are holes in the land (*2d* view) and islands are holes in the water surface (Figure 326). The number of holes in a figure are related to the Betti number, another topological invariant, which is increasingly studied in computational geometry [chazelle, Edelsbrunner].

9. INTUITION AND TOPOLOGY

Some aspects of point set topology are not directly connected to our experience. We think of objects always including their boundary. counterintuitive. In Figure 327 we see a lake with an island: the lake is closed and has a boundary towards land and towards the island. But the island has also a boundary, and island and lake do not overlap! The same difficulty applies to parcels with boundaries (Figure 327)

Figure 326: The Hotel Faakersee on the island in the lake

The intuitive statement that the lake and the parcel do not overlap, in the sense that they do not have points in common is false: lake and parcel overlap, they have the boundary in common. If we remove one of the parcels, then the other parcel and the lake are half-open, they lack the boundary with the

Figure 327: Two parcels on a lake

Figure 328: Cross section of a river

Figure 329: Parcels and lake as close

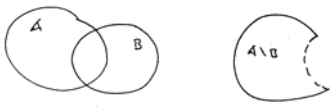


Figure 330: The difference of two sets is half open

An intuitive topology: Closure is applied to all objects.

removed parcel. What are the options to resolve this contradiction?

9.1 PARTIALLY OPEN, PARTIALLY CLOSED OBJECTS

From a position of point set topology, either the parcels are closed and the lake open (does not have a boundary) or the parcel is partially open and the lake closed as shown in Figure 327. This solution is not practical, as all parcels then have to be partially open and the assignment of the boundary to one of the two parcels must be organized (for example, the boundary belongs to the parcel north or west).

Intuitively, we seem to attribute the boundary to the harder object. The river bed (Figure 328) has a boundary, the water in which it floats does not have a cognitively salient boundary; the boundary between the river bed and the water is attributed to the earth, the boundary between river and air above it is attributed to the river. In the composition, the river seems half-open. But when considering the earth, the river or the atmosphere individually, then each is seen as closed.

9.2 ALL OBJECTS ARE CLOSED AND OVERLAP

If both lake and parcels are closed and have boundaries, then they overlap and the overlapping part is the boundary (with an area of zero).This is again counter-intuitive, as the partition of parcels and lakes is made such that they do not overlap.

9.3 PLAUSIBLE ALGEBRA

To capture our intuitions about boundaries better, it is customary to consider all regions as closed and a test for overlap yields a positive answer only when more than just the boundary have points in common (i.e., a non-zero area of overlap). If two objects have just the boundary in common, then we say they 'touch' (for a definition see next chapter).

The result of deducing an object B from another one A, which is the intersection of A with the complement of B, is a partially open, partially closed object A \ B. For a plausible topological algebra, every object is closed; after every operation, closure is applied to all objects. A theory of regular regions is suggested by(Randell, Cui et al. 1992).

10. TOPOLOGICAL CONSTRUCTIONS

From sets *A* with a topology (*A, O*) other topologies can be constructed using the normal methods to construct sets from given sets (subset, sum, and product). The most are:

10.1 SUBSPACE

A subset *B* of the set *A* can be a topological space (*B, O*) with regards the topology *O*.

10.2 SUM

The (direct or disjunctive) sum of two sets is defined as the

$$X + Y = X \times \{0\} \cup Y \times \{1\}$$

If *X* and *Y* are topological spaces, then the direct sum is a topological space, sometimes called the topological sum of *X* and *Y*.

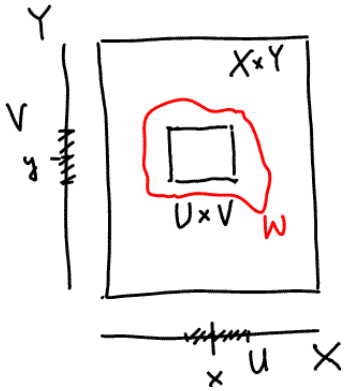


Figure 331: Product space

10.3 (CARTESIAN) PRODUCT

The product of two topological spaces (Figure 331) is a topological space if there are neighborhoods of $(x,y) \in W = X \times Y$, such that *U* is a neighborhood of *x* in *X* and *V* is a neighborhood of *y* in *Y* and $U \times V$ subset *W* {Jänich, 1987 #8849@, 14; Jänich, 2001 #10601; Jänich, 1987 #10602}

11. BASE AND SUBBASE OF A TOPOLOGY

We have seen the importance of the base for a vector space. A similar construction is useful for topology. For example, the boxes $U \times V$ of a product topology form a base.

A base for a topological space *A* is a set *B* of open sets, such that every open set of *A* is a union of open sets from *B*. A subbase *A* is a set *C* of open sets, if every open set of *A* is the union of the intersection of a finite number of sets from *C*. If *X* is a set and *S* a subset of the powerset of *X* then there is exactly one topology for *X* for which *S* is a subbase.

12. SUMMARY

Point set topology is constructed to capture the continuity of space and axiomatizes this notion. It starts with the concept of space as an infinite, continuous set of points. For point set topology, the fundamental concept is neighborhoods, and open and closed sets, for which properties are defined.

Axiomatic definitions of point set topology and topological relations do not produce in constructive solutions. In particular

the influential RCC theory is not constructive, it cannot be implemented directly(Cohn and Hazarika 2001). Point set topology is based on infinite sets, which cannot be implemented directly; for implementation, we need finite representations, which we will achieve with combinatorial topology (see next part xx).

REVIEW QUESTIONS

- What is the essential property of space?
- Define boundary, boundary point?
- What are open and closed sets? Why do they lead to counter-intuitive rules?
- Explain the concept of neighborhood.
- Why is point set topology not directly implementable?
- Draw a simply connected region! Draw one that is not!
- Proof that circles based on the regular Euclidean distance are neighborhoods fulfilling the axioms.

Chapter 22 TOPOLOGICAL RELATIONS

The relations between two objects that are not changed by topological transformations are suitable to describe geographic situations. They are cognitively salient and used in human communication. Natural language terms describe such topological relations (the words *inside* and *overlap* are just two examples); but natural languages do not provide strict definitions for these terms, nor do they have a single, always applicable meaning(Frank and Raubal 1998; Frank and Raubal 1999). This chapter gives definitions for a comprehensive set of topological relations.

Topology determines
metric refines (Egenhofer).

Topological relations play a role in geography and a large variety of definitions and names were proposed for spatial query languages and analysis functions(Frank 1982; Frank, Raper et al. 2001). The goal is to find a function that assigns to every topologically different situation a value describing the topological aspects of the situation. This is not achieved yet; the relations Egenhofer introduced assign to each situation of two simple connected regions with codimension 0 a value, such that this assignment is invariant when the situation is transformed by a topological transformation. There is not yet a function that assigns to each topologically different situation a different value and it is not clear, if such a fine differentiation would be desirable.

The concepts of point set topology are used in a GIS query language, when we ask for all the towns in a county or check, whether Lake Constance is inside Switzerland or at its boundary. Point set topology provides the axiomatization, but is of limited use for implementation.

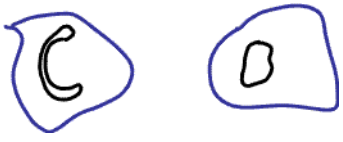


Figure 334: Two island with different shape, but same topology

Topological relations can be composed: If all our knowledge about a situation covers the (topological) relations without knowledge about the metric properties, we still are able to draw interesting conclusions. For example, from knowing that the hotel Faaker See is on the island and the island is inside the Faaker See and the Faker See is in the land of Carinthia, we

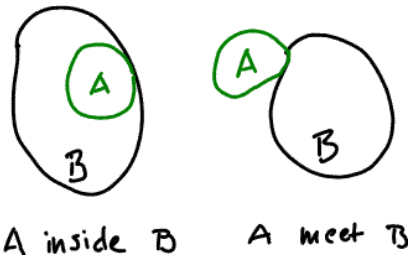


Figure 335: Topological relations in 2d space and

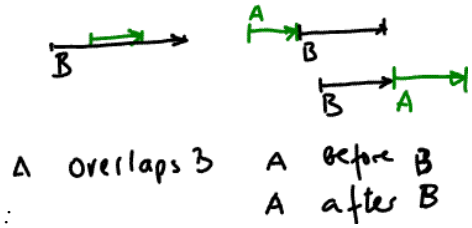


Figure 336: The same topological relations in an ordered domain

Figure 337: (a) simply connected, (b) and (c) are not simply connected regions

immediately conclude that the hotel is in Carinthia (see figure previous chapter xx).

1. INTRODUCTION

Among topological properties relations between objects stand out. Whether two objects touch or overlap or are disjoint influences often what we can do with them. Topological relations are easily observable without measurements and are invariant under topological transformations. There are many ways an island can be inside a lake and many of these situations can be transformed continuously into each other; the functional properties are preserved by these continuous transformations (Figure 334): the island remains an island. It is sufficient to know the topological relation, because it determines the functionality and we need not know anything in particular about the metrics of the situation.

In this chapter we concentrate on the relations between two simple (connected) objects (Figure 335) that are invariant under topologically transformations. The chapter starts with Jordan's curve theorem, which separates objects from their environment. The topological relations were originally investigated for intervals of time by Allen(Allen 1981; Allen 1983; Allen 1984; Allen and Hayes 1985; Allen and Kautz 1985). This investigation of relations between intervals of time mixes the aspect of continuity with order (Figure 336). The subsequent generalization to regions of (unordered) space by Egenhofer discussed the special case of *2d* simply connected regions(Egenhofer 1989). The treatment here separates these two aspects of continuity and order (Figure 335, Figure 336). First, the topological relations for unordered space are discussed before relations between intervals of an ordered domain—for example time, or the z axis, which is strongly ordered by gravity, are discussed.

The purpose of the chapter is to define a function *topRel* that assigns to any two regions in space a single Egenhofer relation. This assignment is invariant under topological transformations.

topRel (a,b) = topRel (f (a), f(b))
where f any topological transformation

Figure 338: Not a figure of geographic interest

The approach is restricted to simply connected (Figure 337) regions with codimension 0. The figures must also be homogenous in the dimension; they must have locally everywhere the same dimension. This precludes figures with spikes, etc. (Figure 338)

2. JORDAN'S CURVE THEOREM

The Jordan curve theorem—probably one of the simplest but most important theorems of geometry—says that a closed curve divides a plane in two regions (and correspondingly for higher dimensions) (Figure 339). From the curve theorem follows, that a line connecting and interior point of a region with an exterior point crosses the boundary line.

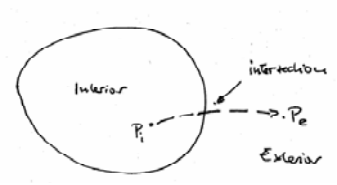


Figure 339: A line connecting an interior point with an exterior point must cross the boundary

3. TOPOLOGICAL RELATIONS BASED ON SET OPERATIONS ONLY

The relations between two regions obtained by the set operations are invariant under continuous transformations. One could just separate *disjoint*, *intersect*, *equal*, *inside*, and *contains*/inside.

- A *disjoint* B: $A \cap B = 0$
- A *intersect* B: $A \cap B \neq 0$, $A \cap -B \neq 0$, $-A \cap B \neq 0$
- A *equal* B: $A \cap B = A$, $B \cap A = B$
- A *inside* B: $A \subseteq B$
- A *contains* B, B *inside* A: $B \subseteq A$

Contains is the converse of *inside*, whereas *disjoint* and *equal* are symmetric.

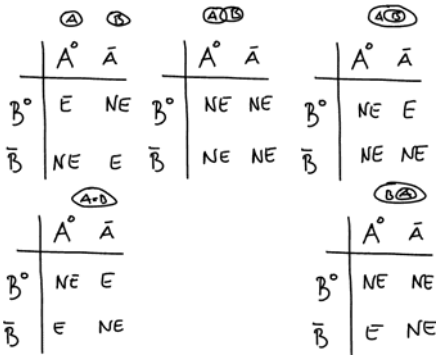


Figure 340: The characterization of the five relations differentiated with set operations

These relations do not cover all the relations that people differentiate and that are relevant for the functioning of objects. It is, for example, not possible to differentiate between general *disjoint* and the special case of being the *neighbor*; Zurich and Rapperswil are on the Lake Zurich, but Schwyz and Uster are not (Figure 341). It is to differentiate such situations and the topological relations Egenhofer differentiates achieve this:

4. TOPOLOGICAL RELATIONS FOR SIMPLY CONNECTED REGIONS

For simple connected regions (Figure 337) in any unordered space, 8 relations can be separated: *disjoint*, *touch*, *overlap*, *covers* (with the converse *covered by*), *inside* (with the converse *contains*) and *equal*. The semantics of these 8 terms are not the same as those with the same name defined by set operations (see section 3 above). They were formally defined by Egenhofer in

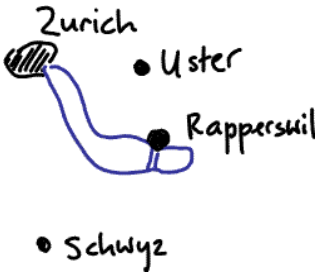


Figure 341: Relations of cities and a lake

his Ph.D. thesis (Egenhofer 1989) and later included in the spatial extension of the SQL standard (Egenhofer used the term "meet", which I replace here with "touch" to avoid confusion with the lattice operation *meet*). These relations are independent of the dimension of the unordered domain and require only that the regions are simply connected (Figure 337).

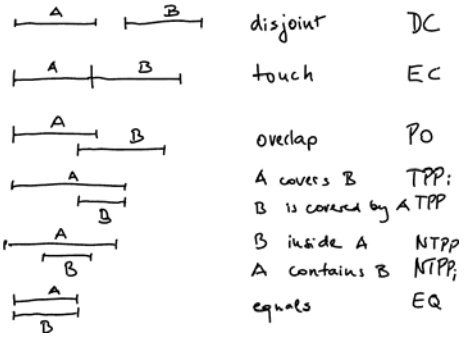


Figure 342: The eight topological relations in an unordered domain (the last column gives the names used in the RCC calculus)

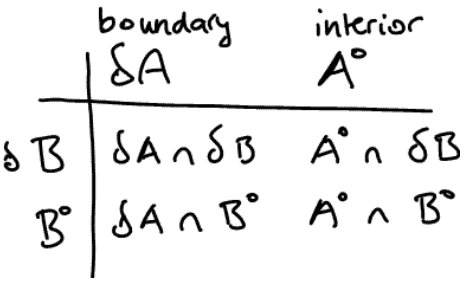


Figure 343: The four intersections for boundary and interior

4.1 Topological relations of simply connected objects with co-dimension 0: the 4 intersections

Egenhofer observed that topological relations between simple-connected areas can be expressed in terms of the intersection of the interiors and the boundaries of the two objects; it is sufficient to observe, whether these intersections are empty or non-empty (Figure 343).

Relations characterized in this way are certainly invariant under topological transformation: topological transformation must transform the interior of a region into the interior of the transformed region and the boundary into the boundary of the transformed region. The four characterizations of a relation between two regions are the same before and after the transformation:

For simply connected objects with co-dimension 0 it is possible to differentiate between topological relations by considering the pairwise intersection of boundaries and interiors and testing only for emptiness or non-emptiness. This gives a total of $4^{2 =}$ 16 different combinations of boundary or interior that can be intersected and the result is either empty or non empty. From the total of 16 different combinations, only 8 can be realized with simply connected figures with codimension 0(Egenhofer 1989).

In Figure 344 and Figure 345 the eight possible topological relations geometric configurations are shown, the termini Egenhofer proposed (with *touches* for Egenhofer's relation *meet*) and the defining 4 intersection values are given as empty (E) or non-empty (NE).

Four relations are symmetric: *rel A B => rel B A* . The matrix for the intersection values for these relations must be symmetric. They are: *disjoint*, *meet*, *overlap*, and *equal*. Four relations are non-symmetric: *inside* and *covers*. They have a converse relation, namely *contains* and *covered by*.

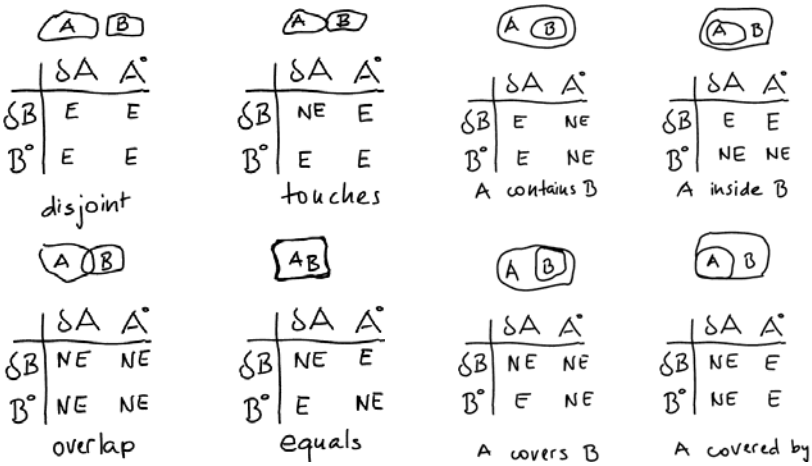


Figure 344: The 4 symmetric relations *Figure 345: The four non-symmetric relations*

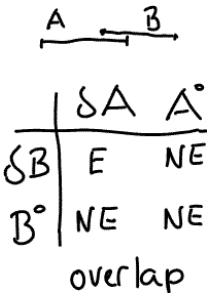


Figure 346: Overlap in 1-d

The relation *overlap* is special for the *1d* case: the intersection of the two boundaries is empty (Figure 346), the entry for *overlap* in Figure 344 is valid for *d >= 2*.

4.2 Classification based on the Connected property

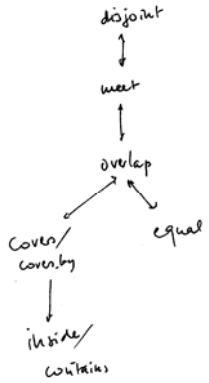
At about the same time as Egenhofer proposed the classification of topological relations based on the intersection of boundaries and interiors, Cohn and co-workers proposed a classification based on a single predicate *connected*. Their RCC contains different definitions for the same 8 relations between simply connected regions(Cohn 1995).

This second definition of the same topological relations confirms the fundamental nature of the relations identified by Egenhofer. Egenhofer's approach is based on the intersection of boundary, interior and, later, exterior of the regions connects better to combinatorial topology (see later chapter 24xx) and is directly implementable, whereas the RCC axiomatization is based on point set topology and difficult to use as guidance for implementation; technically we say that the RCC calculus is not constructive(Cohn and Hazarika 2001).

5. Conceptual Neighborhood for Topological Relations

The 8 topological relations can be arranged in a succession of relations that are obtained if one figure is moved with respect to the other figure (Figure 349, The relations that can be differentiated with set operations are different from the Egenhofer relations, even if they have the same names. For them, a conceptual neighborhood graph can be drawn (Figure 348); it is similar, but contains less relations.

). Freksa proposes to call such diagrams conceptual neighborhoods(Freksa and Mark 1999). Consider two figures of unequal size: First the two figures are *disjoint*, then they *touch*, *overlap*, *cover*, are *inside* (if the two figures of equal size, then the succession is *disjoint, meet, overlap, equal*). Relations *equal*, *touches* and *covers/coversBy* are *dominant(Galton 1997)*; they hold only for one instant, whereas the other relations hold for many different positions.



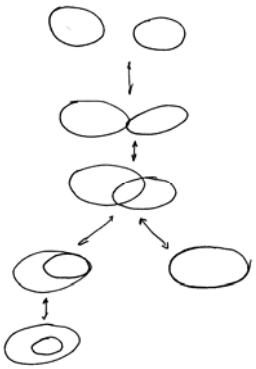


Figure 349: Conceptual neighborhood

The relations that can be differentiated with set operations are different from the Egenhofer relations, even if they have the same names. For them, a conceptual neighborhood graph can be drawn (Figure 348); it is similar, but contains less relations.

Figure 347: Conceptual neighborhood graphical

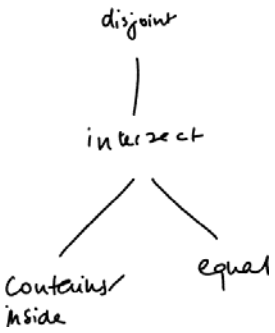


Figure 348: Conceptual neighborhood for set operation based relations

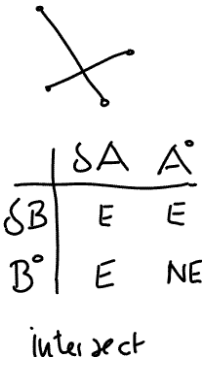


Figure 350: Intersection of two line segments(codim = 1)

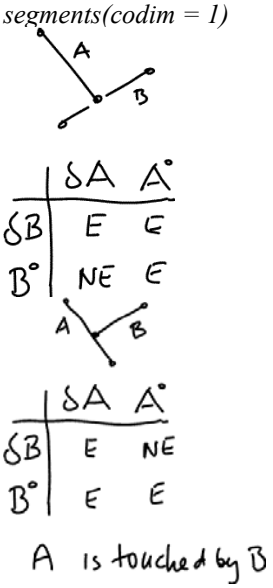


Figure 351: Two line segments touch-1

6. EXTENSIONS OF THE *FOUR INTERSECTION* TOPOLOGICAL RELATIONS

From Egenhofer's thesis sprang a rich literature discussing extensions to the base set of topological relations. There are clearly topologically different situations that have the same Egenhofer four-intersection values (Figure 352) and suggestion how to separate them abound.

6.1 RELATIONS BETWEEN OBJECTS WITH CO-DIMENSION NOT ZERO

The original definitions by Egenhofer are valid for simple-connected, *2d* regions in *2d* space. For lines in *2d* space, which have co-dimension *1*, topologically different situations can not be differentiated (Figure 352). For 2 simple lines, the four intersection method gives one more symmetric relation: intersect (Figure 350) and a non-symmetric one (touches-1, touchedBy-1, (Figure 351). These relations are topologically invariant, but

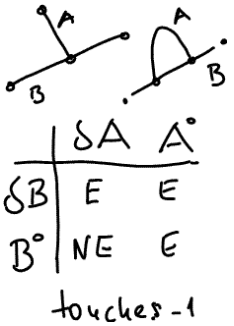


Figure 352: Two distinct configuration result in the same Egenhofer relation

Figure 353: Touches'

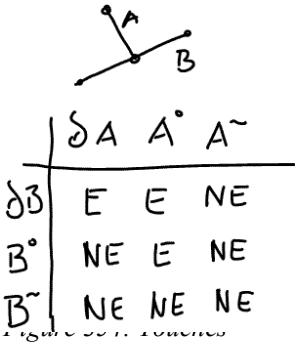


Figure 354: Touches"

configurations that are topologically distinct result in the same relations; they are not sufficient to differentiate all topologically distinct situations (Figure 352).

6.2 NINE INTERSECTION MODEL OF TOPOLOGICAL RELATIONS

To differentiate the two situations in Figure 352 observe not only the intersections between boundaries and interiors, but also the intersection between them and the exteriors of the figures. This is called the nine-intersection model, because the intersection between boundary, interior and exterior are checked; this gives 9 intersections, which are tested for empty or not(Egenhofer and Herring 1991).

The two relations touches' and touches" can be distinguished in the nine-intersection model (Figure 353, Figure 354). The nine-intersection model is a superset of the four-intersection model and distinguishes between simple connected regions with co-dimension 0 the same relations than the four-intersection model. There are $2^9 = 512$ relations possible, but most of them cannot be realized with simple connected objects.

There are a total of 33 topologically distinct relations between two simple lines and 24 additional ones between non-simple lines (Figure 356). Egenhofer and Herring have also identified 20 relations in *2d* space between a region (without hole; Figure 355) and a line(Egenhofer and Herring 1991). Ten relations exist between two regions with holes in addition to the 8 that exist also between regions without holes. Even with these fine distinctions, which are much finer than what natural languages has simple terminology for, nine-intersection can still not differentiate all topologically different situations (for example the two situations in Figure 359 are not distinguished). It seems not possible to achieve such a definition for topological relations..

The topological relations assign to two regions an Egenhofer relation, such that any topological transformation of the two regions gives the same relation. Two situations *(A,B*) and *(C,D)* that result in the same topological relations *r* (based on the 4 or 9 intersection model) *r = topRel (A, B) = topRel (C, D)*, can be topologically distinct, i.e. there is no topological transformation *g*, such that *g (A) = C and g(B) = D*. This means: Two pairs of regions which result in the same Egenhofer relation are not always topologically equivalent (Figure 358).

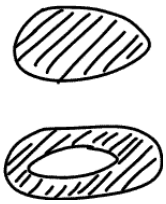


Figure 355: A region with and without hole; with a connected or disconnected boundary

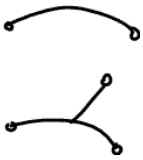


Figure 356: A simple and a complex line

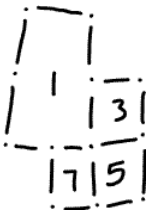


Figure 357: Parcels

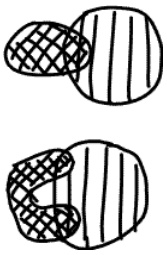


Figure 358: Two pairs of regions which both overlap but are not topologically equivalent

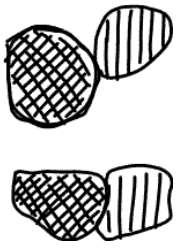


Figure 359: Two situations which are only differentiated by the dimension of the intersection

6.3 *Refinements of the topological properties*

Egenhofer and Franzosa have identified a number of refinements for the intersection of boundary and interior, which distinguishes the situation. For each intersection it is not only tested if it is empty or non-empty, but for non-empty intersections we distinguish:

- the *number* of components (in Figure 358 the number of components of interior-interior intersection is in the upper figure is *1* and in the lower figure is *2*)
- the *dimension* of the component is identified (in Figure 358, the upper figure has dimension *0* for the intersection of the two boundaries, the lower figure has dimension *1*)
- the *sequence* of boundary-boundary intersections or the types of boundary-boundary intersections(Egenhofer and Franzosa 1995).

The distinction of intersection of different dimension helps to distinguish between two types of neighbors. Consider the four parcels in Figure 357: both 1 and 3 and 1 and 5 are *touching*— are 3 and 5 both neighbors of 1, and with what definition? We will later call the 1 – 3 relationship a 4-neighborhood and 1 – 5 an 8-neighborhood (see xx).

6.4 *Holes*

Regions with holes, which were excluded in the initial definitions by Egenhofer, pose additional problems for the classification of topological relations: Figure 360 gives two situations that give relation *overlap*, but are topologically distinct. A region with holes can be seen as a region without the holes plus a collection of holes. The topological relations between n regions (without holes) are completely specified by n^2 relations. Between two regions with m and n holes exists a total of $(m + n + 2)^2$ topological relations describing the configuration. Some relations are redundant and only $m*n + m + n + 1$ are necessary(Egenhofer, Clementini et al. 1994).

6.5 *Combining Topological Relations with metric relations*

The topological relations relate to an order relations between simply connected regions:

A inside B implies (size A) < (size B)
A equal B implies (size A) = (size B)

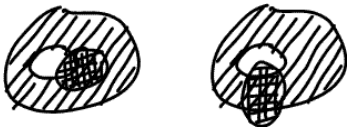


Figure 360: Two topologically different pairs of regions, which cannot be distinguished by the 4-intersection

	∂A	A°
∂B	NE	NE
B°	NE	NE

Figure 361: The 4 intersection for both pairs in Figure 360

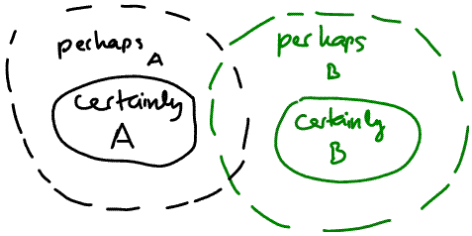


Figure 362:An Egg-Yolk representation of two objects with uncertain boundaries: "perhaps overlap" (Cohn and Gotts 1996)

Figure 363:(size A) < (size B) but not A inside B

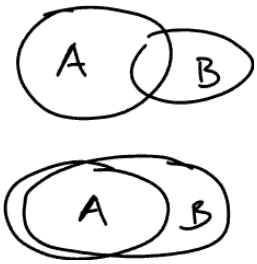


Figure 364: Two different gestalt of overlap

Inside imposes a partial order, whereas size a total order: from (size A) < (size B) does not follow that A inside B (Figure 363).

It is tempting to record in the values of the intersection matrix an indication how much they intersect. For example, two areas that *overlap* a little bit and two areas that overlap a lot have both the same topological relation, but the Gestalt is different (Figure 364). The same applies for *touching* and the other topological relations. The measure of how much the two boundaries or interiors overlap, is best expressed as a ratio of the overlap, which makes the measure independent of the size of the objects. Such relations are not topological because they are not invariant under a continuous transformation, but they are expressing useful properties in a scale independent manner.

6.6 *Topological relations and approximations*

Objects with indetermined boundaries (Burrough and Frank 1995; Burrough and Frank 1996) can be represented by an area that certainly is occupied by the object and a second area where perhaps the object is situated (Figure 362). One might ask what are the topological relations between two objects represented in this way? What relations obtain certainly? What relations may obtain? This has been explored but applications are not yet using these distinctions(Clementini and Di Felice 1996; Cohn and Gotts 1996).

7. Cognitive Plausible Topological Relations

The differentiations in 8 topological relations covered by the four intersection model cover a large part of the practically relevant cases(Mark and Egenhofer 1992; Egenhofer, Sharma et al. 1993; Egenhofer and Mark 1995; Shariff, Egenhofer et al. 1998). Mark and Egenhofer have checked with extensive subject tests that the topological relations differentiated by intersection of interior, exterior, and boundary are cognitively plausible: they are the kind of relations people differentiate(Mark and Egenhofer 1992; Egenhofer and Mark 1995).

8. Allen's Relations between Intervals in Time

Allen (Allen) has classified the relations between time intervals into 13 named relations (Figure 365). Time intervals are simply connected regions of an *oriented* line. Oriented intervals have a start and an ending point that is differentiated. This leads to a finer distinction of the relations, where relations that are

symmetric in an unordered domain become antisymmetric in the ordered domain:

- *Disjoint* becomes either *before* or *after*
- *Touches* becomes *meets* or *met-by*
- *Overlap* becomes *overlaps* or *overlapped-by,* etc.

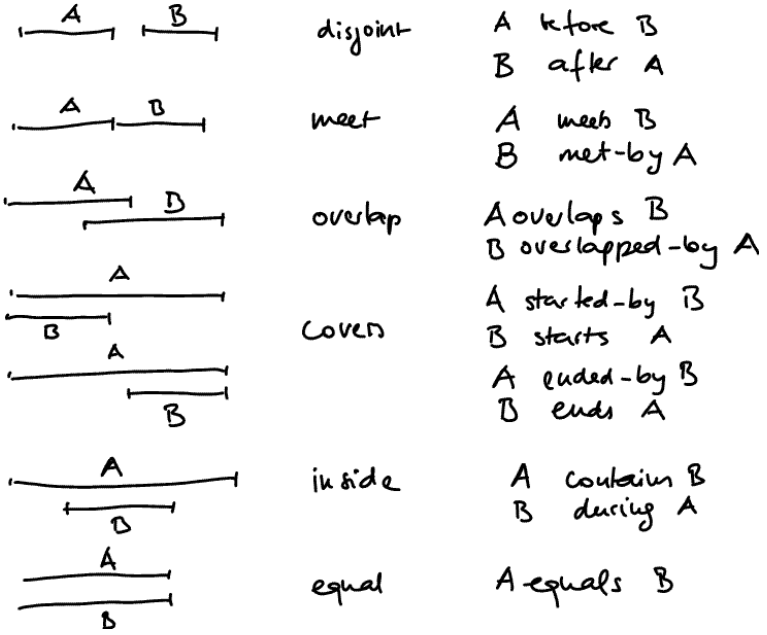


Figure 365: The relations which are differentiated between simply connected regions of an ordered domain

9. Generalization to Topological Relations in Ordered *N*-Dimensional Spaces

Allen's characterization of relations between temporal intervals can be derived from the Egenhofer relations and a differentiation of order. It is customary to express this as conditions on the starting and ending points:

starts i j = (start i == start j) && (finish i < finish j).

It may be simpler to determine the center of gravity (the zero moment, see xx) of the two intervals and classify first using the four intersection model and then select the precise temporal relation:

A before B = A disjoint B && center A < center B
A start B = A covers B && center A < center B

Figure 366: Touching southward

Figure 367: Touching eastward

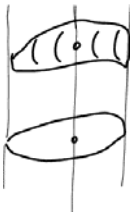


Figure 368: Projection of interior is interior of projection

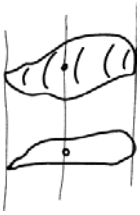


Figure 369: Projection of boundary is not boundary of projection

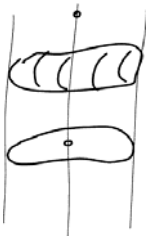
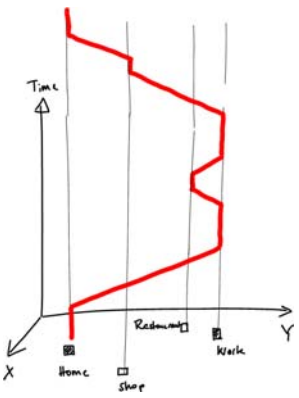


Figure 370: Projection of exterior is not exterior of projection



etc.

This approach is generally useful in *2d* (general *n-d*) spaces. In an ordered *2d* space, we can find one or two ordered axis. Geographic space, for example, is ordered by the north-south direction, which influences the amount of sunshine available, and on the east-west direction or an inland-sea direction. Italy *touches* Switzerland, but this is mainly in the north-south direction, whereas the relation between France and Germany is a touching in east-west direction (Figure 366 and Figure 367). The same qualifications can be applied to covers, contains etc.: Styria is south *inside* in Austria (earlier figure xx).

10. PROJECTIONS AND TOPOLOGICAL RELATIONS

Most GIS represent objects with the *2d* projection in the *x-y*-plane. Unfortunately, the topological relations between two regions in *3d* space and the *2d* projections are not simple and I know of no coherent theory. The problem is that the projection of the boundary of a *3d* object is not the boundary of its *2d* projection (Figure 369), the projection of the interior *is* the interior of the projection (Figure 368), but the projection of the exterior is not the exterior of the projection (Figure 370).

Similar Projections occur when we consider space and time. Hagerstrand has introduced the Space-Time diagram into geographic research, where the projection of the location of a moving object is combined with time shown on the *z*-axis (Figure 371). In such space-time diagrams we can show as cones all the points a person can possibly reach when we know her location at a given time (Figure 372). One may then ask if two people can possibly have met, i.e., if their cones of reachability have intersected(Hornsby and Egenhofer 1997).

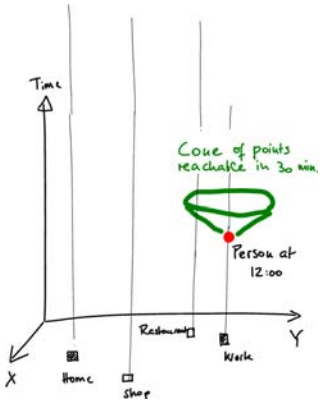


Figure 371: A person's trip from home to work, lunch at a restaurant and a stop at a shop on his way home

Figure 372: Cone of points reachable in 30 minutes

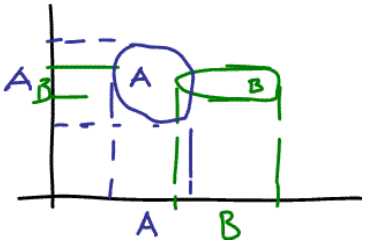


Figure 373: Symbolic projection of A and B

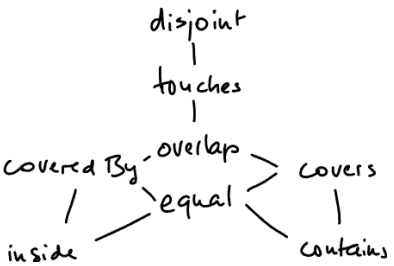


Figure 374: Transition by move or change of region

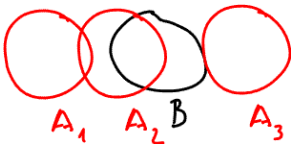


Figure 376: A moving object A at three different times before the fixed object B

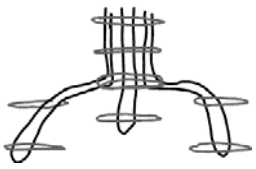


Figure 375: Pattern for translation

11. Symbolic Projection

An earlier approach to classify relations between objects was using on *symbolic projections(Chang, Jungert et al. 1989; Jungert and Chang 1989; Chang, Jungert et al. 1990; Jungert 1992; Jungert 1993)*. It characterized the relations of regions by the relations of their projections to the coordinate axis (Figure 373). This approach was designed for searching in a large database of images, because each image can be represented by two strings and search for substrings is fast. The major disadvantage is that this approach is not invariant under rotation.

12. Moving and Changing Regions

Egenhofer's topological relations and especially the conceptual neighborhood can be understood as the sequence of relations between a fixed and a moving object (Figure 376). A sequence of topological relations is characteristic for certain types of changes in regions. In the conceptual neighborhood graph different movements result in different transitions between two relations that can occur without any other intervening relations(Egenhofer and Al-Taha 1992). This can be used, to differentiate different types of movements —rotations or translations—and changes to the region—growing or shrinking—as distinct patterns of changes (Figure 375, Figure 377, Figure 378, and Figure 379):

13. Conclusion

A cognitively plausible set of topological relations has been identified and formally defined by Egenhofer(1989). The relations are differentiated by overlap of interior or boundary of the objects. To compute these relations from geometric objects represented in a GIS, it is necessary to be able to compute the interior and boundary of the objects and to test these for intersection or emptiness of intersection.

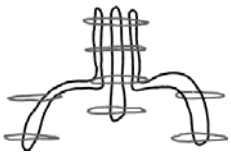


Figure 377: Pattern for rotation

Figure 378: Pattern for expanding A or shrinking B

Figure 379: Pattern for shrinking A or expanding B

Figure 380: Components A touch components of B (from south-east)

These topological relations are invariant under continuous transformation *f*:

topRel A B = topRel (f A) (f B)

but not all topological separable relations result in different Egenhofer relations. Egenhofer's approach differentiates originally between 16 relations (2^4), of which not all can be realized in *1d* or *2d* space for simple connected figures. For simple-connected 2d regions in 2d space, only 8 are possible for objects with codimension 0.

Egenhofer relations are restricted to simple connected figures. The generalization to consider the 9 intersection between interior, boundary, and exterior of the figure can differentiate more relations; refinements are possible if the dimension of the intersection, the number of components or even the size of the intersection is considered. The resulting large number of different topological relations may be useful in some applications, it seems not to lead to a generally useful, cognitively manageable set of relations.

For objects with multiple components or holes, a matrix that gives the relations between all the components describe the relation; again the analysis of such situations is application dependent. For example, it may be interesting for some applications to observe that the components of object A and object B pairwise overlap, or pairwise touch (Figure 380).

In ordered domains, some relations can be differentiated by observing the relation of the center of gravity of the two objects. For time, Allen's differentiated 13 relations between two intervals. This can be seen as the product of Egenhofer's relations and the order of the center of gravity.

14. Review Questions

- What are the topological relations defined by Egenhofer? How are they defined? Why are exactly these relations differentiated?
- What does it mean to say that a relation is dominant? Which relations are dominant?
- What relations can be differentiated when only interior and exterior are considered?
- What is a ‘conceptual neighborhood’?
- What is the difference between 4 and 9 intersection relations?

- What are space-time diagrams? Draw one for your trip from home to the university.
- Why do symmetric relations not have a converse?
- Build the table of distances between the relations in the four intersection tables (count each difference in an entry as 1 unit distance); what are the connections with the smallest distance? How to interpret?

PART EIGHT ALGEBRAIC TOPOLOGY: SIMPLEX AND COMPLEX

Combinatorial topology is approaching topological problems with algebraic methods. In part 6 the geometry of unbounded geometric objects (flats) was clarified and in part 7 topology and how it structures the continuous space of bounded objects were shown. The three chapters in this part use combinatorial or algebraic topology to deal with geometry of finite objects: segments of infinite lines, triangles, but also regions of arbitrary shape, and eventually, subdivisions of space. A GIS is using these structures to record the shape and the position of all the things it collects information about: parcel and their boundaries, lakes and woods but also street networks or the gas distribution pipes.

The goal of the chapter is a method to calculate the intersection of two arbitrary figures, independent of the dimension or complexity.

Combinatorial topology is based on *counting* of finite objects(Henle 1994, 5). Countable, finite objects are more amenable to implementation on computers than sets of infinite number of points in point set topology. What are the geometric objects that can be counted to capture the notion of continuous space? The well-known Euler characteristic for a cell is a prime example for a count, which is invariant under topological transformations (Figure 381): $F - E + V = 1$, where F is the number of faces, E the number of edges and V the number of vertices.

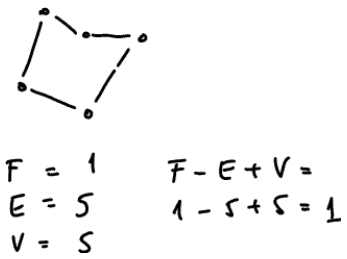


Figure 381: Euler's polyeder formula for a cell

Combinatorial topology studies invariants of bounded geometric objects under topological transformations. We first investigate the simplest geometric configurations: points, straight line segments, triangles and operations applicable to them. These simplices have great advantages. They are convex parts of flats and have a fixed number of boundary points for each dimension.

Simplex:
part of flat
convex
fixed number of boundaries

The second chapter constructs complex geometric objects from simplices. The purpose is to construct an algebra that is closed under intersection and union. The third chapter shows how metric information is used to construct the simplicial complex necessary. This completes the goal of this part: a

general method to intersect arbitrary figures results from the merging of the cell complexes of the given figures.

Simplicial complexes have several applications in GIS: they are a generalization of graphs, which will be treated in the following part, and subdivisions, especially triangulations (part 10).

Chapter 23 GEOMETRIC PRIMITIVES: SIMPLICES

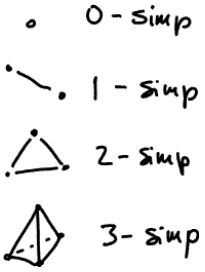


Figure 382: Simplices (they are open and do not include the boundaries!)

Geometric objects are composed of points, lines, and areas. We study in this context the simplest forms, called simplices (Figure 382). They form the building blocks from which more complex objects are then constructed in the next chapter.

Union and intersection is not computable for simplexes (Figure 383). This chapter is therefore laying the ground for the discussion of simplicial complexes in the next chapter, where eventually merging complexes allows union and intersection of arbitrary complexes (chapter 25xx).

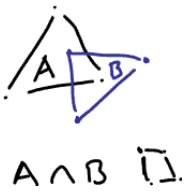


Figure 383: The intersection of two simplices is not a simplex!

1. INTRODUCTION

The vector algebra (chapter 9xx) introduced geometric lines of infinite length, which was generalized to the notion of flats (chapter 19xx). In this chapter we construct geometric values for the simplest geometric configurations with *finite* geometries: *points*, straight *line* segments, *triangles*, etc. There is an simplex for any dimension, generally described as *n*-simplex (or *n*-simp).

Similar to previous chapters, the focus is on the general case, the *n*-simplices. Dimension of objects is decisive: we have different words for properties of depending on their dimension (length, area, volume). This is traditional approach for computational geometry, which goes back to Hilbert's approach to geometry, applies also to GIS and CAD; it is cognitively well justified, but has not lead to a consistent and attractive theory.

In this chapter, different geometric objects—points, lines, and triangles—are generalized to a single class simplex (*n*-simp) with operations that apply to all of them. Object-oriented software engineering (Wegner 1987; Meyer 1988; Rumbaugh, Michael Blacha et al. 1990; Egenhofer and Frank 1992) calls this generalization: the common operations applicable to several classes are identified and described at a general level. This dimension independent approach continues the dimension independent treatment of flats and the discussion of topology.

2. SIMPLEX DEFINITION

Simplices are the simplest finite, open geometric figure in space of any dimension (Figure 382). They are constructed from the least number of simplices of a lower dimension.

0 simp – point,
1 simp – line,
2 simp – triangle,
3 simp – tetrahedron, etc.

Simplices are the object of studies in combinatorial (or algebraic) topology; for their use in GIS, we must consider their metric aspects as well: their properties, the relations between simplices and the operations applicable to them can be separated in metric and topological ones. Topological properties are:

- Rank, dimension, codimension, and corank
- Orientation

Rank = dim + 1
A simplex of rank n is spanned by n points.

Metric properties are:

- Length, area, volume, generalized to size, moments
- Distance
- Orientation.

Relations between simplices are:

- Equality
- Incidence and adjacency relations,
- Boundary relation and the converse co-boundary.

Operations applicable to simplices are:

- Join, to construct a simplex from points,
- Reverse a simplex,
- Intersection test,
- Point in simplex test,

3. TOPOLOGICAL VIEW OF SIMPLEX

A simplex is the image (under continuous transformation) of the unit sphere of the appropriate dimension (Figure 384). The sphere of *2*-dimension is a circular disc, the sphere of dimension *3* is a ball—higher dimensions are somewhat more difficult to imagine, but they are topologically equivalent to the image of the product of *n* unit intervals (0..1) (*n*= number of dimension). The unit sphere is part of the flat of the corresponding dimension and the simplex is imbedded in the flat of the corresponding dimension.

Figure 384: Simplex is topological image of unit sphere or product of unit intervals

Figure 385: The body of s a 2-simplex: the 2-simp, three 1-simps and three 0-simps

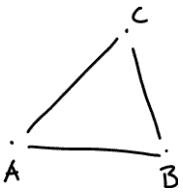


Figure 386: Construction of simplex as join of points

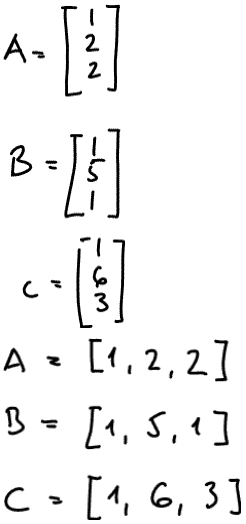


Figure 387: The construction of a 3-simp from 3 points

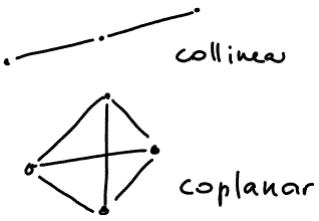


Figure 388: 3 collinear points are not a triangle, 4 coplanar points are not a tetrahedron

det (join a_1, a_2,, .. a_n) = 0 <=> a_1, a_2,, .. a_n not in general position

Figure 389: Join a point to a line gives triangle

The simplices are open, they do not include their boundaries. The closure of a simplex is called the body of the simplex; it includes the simplex and all its boundaries (Figure 385).

4. A Simplex Results from Joining of Simpler Simplices

Simplices are constructed form a number of points in most general position (not collinear, not coplanar) (Figure 386). One point gives a *0*-simplex, two points give a *1*-simplex, etc. (Figure 382). We consider here only oriented simplices, for which the order of points is significant. They are constructed from oriented flats (see chapter 19xx). The operations join used to construct is the same as in chapter 19 and not commutative (as is it usually in lattice theory).

join a b = reverse (join b a).

Note: the coordinates are expressed in homogenous coordinates.

The join operation is the same as used to construct flats: the result of the join is the matrix of the (column) vectors of the points. The matrix construction maintains the order of the points. The points must be in general position,. If the points are not in general position the determinant of the resulting matrix is 0; this is a convenient test to identify degenerated simplices (Figure 388).

Join is generalized and takes not only points as inputs, but simplices of any dimension can be joined (this is similar to the composition of flats to flats of higher dimension). Joining a point with a *1*-simp gives a *2*-simp, etc. (Figure 389).

The inverse operation to join is *0-skeleton*: it gives for each simplex a list of the 0-simplices it is constructed from.

5. Dimension, Rank,

It is convenient to define the **rank** of a simplex as its dimension +1. The rank of a simplex is the number of points it is constructed from, i.e., the rank of a simplex is *card . skeleton*. A simplex is embedded in a **flat**—this is called the span of the simplex. The dimension of the simplex and the dimension of the **span** is the same.

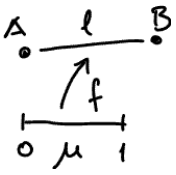


*Figure 390: l is f μ, with f μ = a + μ * (b-a)*

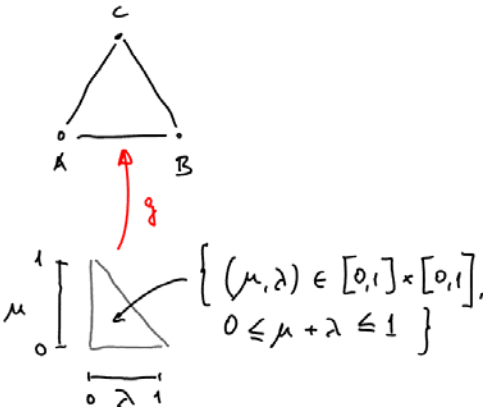


*Figure 391: Triangle is g(μ , λ) = a + μ * (b - a) + λ * (c - a)*

The rank is dimension plus one and is always >= 0.

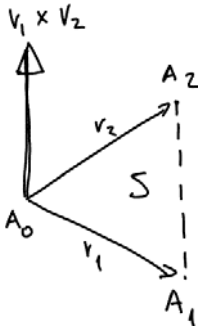


Figure 392: The positive arrow on a simplex S

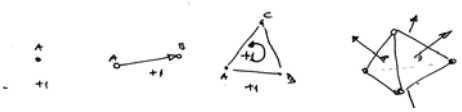


Figure 393: Simplices with positive orientation

The dimension indicates how many *degrees of freedom* or how many parameters are necessary to describe such a figure (Figure 391). The points of the simplex define vectors, which can be seen as bases for a space. The dimension of this space is the dimension of the matrix formed from the points of the simplex. For an *n*-simplex with points $a_0, a_1, .. a_n$ the vectors a_1-a_0, a_2-a_0, ... a_n-a_0 are linearly independent and define the span of the *n*-simplex (note: $a_0, ... a_n$ are not linearly independent!).

For regular simplices dimension is 0 or a positive integer; it will be written as *n*-simp. By convention the empty set is defined by 0 points, has rank 0 and dimension -1.

6. CO-DIMENSION

Simplices are embedded in geometric space. The codimension of a simplex is the difference between the dimension of the embedding space and the dimension of the simplex. A point in *3d* space has codimension 3, whereas a line in *2d* space has codimension 1. Co-dimension is 0 or positive integer. The same as for flats, the co-dimension is the same as the co-rank.

Codim (dim space) – (dim simp) = codim (dim space) – (rank simp) + 1
codim = corank

7. ORIENTATION

The orientation of simplices follows from the orientation of the flats, which we constructed as oriented. A simplex embedded in an oriented flat inherits the orientation. The orientation of a simplex is either positive or negative (represented by the values +1 or –1). The direction of a line is either from point *A* to point *B* or the reverse. Areas have two directions as well, defined as the sense in which the points on the circumference are listed: anti-clockwise is positive, clockwise is negative (Figure 392, Figure 393). By convention, a *0*-simplex has positive orientation.

Volumes have two orientations as well. The orientation of a volume is defined positive if the vectors resulting from the cross product of the edges point all outwards,; if they point inside, then the volume has negative orientation. Areas and volumes with negative orientation can be interpreted as holes in an area or volume of positive orientation (Figure 394).

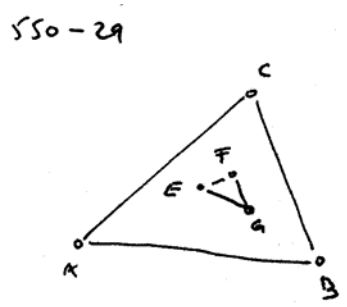


Figure 394: Triangle ABC with hole EFG

The orientation in the projective representation is taken as the sign of the determinant of the homogenous matrix that defines the simplex (function clockwise respective counterclockwise). This is the counterclockwise (CCW) predicate in chapter 20.

orientation (s_0, s_1,.. s_n) = sig (det (join (s_0, s_1..., s_n))

The sign of the determinant is the orientation
sig x = if x > 0 then +1 else
if x = 0 then 0 else -1

8. EQUALITY OF SIMPLEX: PERMUTATIONS OF BOUNDARY REVERSE SIMPLEX

Two simplices are equal if they consist of the same points in the same order, but this is too restrictive a definition of equality: the triangles (ABC), (BCA), and (CBA) are the same.

Two matrices describing a simplex change only the sign of the determinant if two rows are columns are exchanged, called a transposition (see chapter 10). This applies to simplices, which can be seen as a topological image of the simplex spanned by the unit vectors transformed by a matrix(Stolfi 1991 192).

Two simplices are equal if they consist of the same points and the order of the points can be transformed from one to the other with an *even* number of exchanges between two points. Cyclic permutations of n element correspond to *n-1* transpositions. A cyclic permutation of a *2*-simp gives the simplex with the reversed orientation. AB is the reverse of BA. A cyclic permutation of a *3*-simp does not affect the simplex. A B C -> C B A -> B C A are all the same simplex. (Stolfi 1991, 192)

Cyclic permutation of 2-simplex does not change; cyclic permutation of 1-simplex does change the orientation.

9. BOUNDARY

The boundary of a simplex has a dimension of one less than the dimension of the simplex: the boundary of a line (*1d*) is two points (*0d*). The boundary of a triangle (*2d*) is the three lines (*1d*). The boundary of a point is the empty set. The dimension of the boundary is one less than the dimension of the simplex (for consistency, the dimension of the empty set is defined as *-1*).

Attention: boundary for simplex is not the same as boundary in point set topology. Take a *2*-simp (Figure 385), the boundary are the 3 *1*-simps and does not include the *0*-simps at the corners! The point set topological boundary includes the lines and the corner points.

Boundary is a relation between a simplex of dimension *n* and simplices of dimension *n-1*. It has a converse relation, co-boundary, which will be used in the next chapter on complexes.

From the boundary relation follows the *boundary* function, which takes a simplex and returns the set of boundary simplices.

```
boundary :: simplex -> [simplex]
```

10. METRIC OPERATION: LENGTH, AREA, VOLUME, ETC.

All simplices have a size: *1*-simplex have a length, *2*-simplex an area, *3*-simplex a volume. For *n* simplices (rank *n+1*) with rank >=the volume is the value of the determinant of the (*n+1*) vectors in homogenous (*n+1*) coordinates, corrected for the product of the homogenous values and divided by the factorial of *n* (*n*!).

$$\text{vol}\ |A_0, A_1 .. A_n| = \frac{1}{n!} \frac{\det [A_0, A_1, A_n]}{w_0 \cdot w_1 \cdot w_2 .. w_n}$$

The size of a simplex of dimension *0* (point) is assumed to be *0*. The size of a *1*-simp is the length, computed as the norm of the difference of the two vectors. If the points are given in ordinary (not homogenous) coordinates, then the volume is determinant of the matrix formed by subtracting one of the points from all the others:

$vol\ [A_0, A_1, .. A_n] = (1/n!) * det\ |A_1 - A_0, A_2 - A_0, .. A_n - A_0|$.

The computation of the size yields a signed quantity that gives also the orientation; "ordinary" size is the absolute value, expressed as a real (and approximated by a Float). One might think of the area as the absolute value of the determinant, and think of negatively oriented simplices as holes!

```
orientation = sign vol
```

11. TEST FOR POINT IN SIMPLEX

The test whether a point is inside a simplex or not is the base operation for the general intersection test for simplices. It is useful to separate the case, where the given simplex has codimension 0 or not.

11.1 POINT IN SIMPLEX WITH CODIMENSION 0

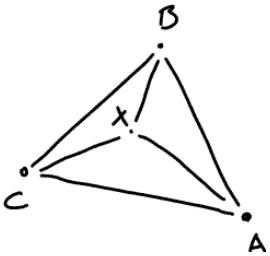


Figure 395: X is inside

Take the example of a triangle (Figure 395) and test three times if the new point is left of the *1*-simp, which means apply three time the counterclockwise (*ccw*) predicate (chapter 19): *X* is inside if it is left of *AB*, *BC* and *CA*, which means *ccw (ABX)* and *ccw (BCX)* and *ccw (CAX)* must all be positive.

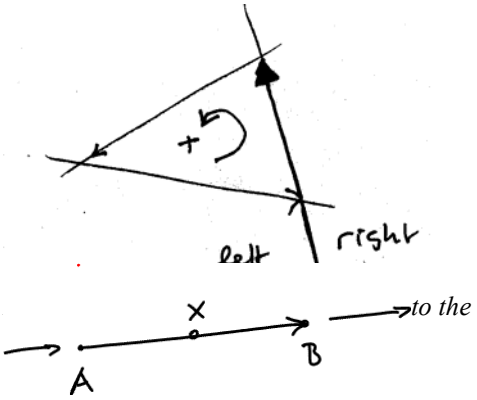


Figure 397: X is inside AB (1D space)

Note: to properly deal with degenerated simplices or points lying on the boundary, the CCW test must yield a result of (+1, 0, -1} which leads to a three valued logic(Sinowjew 1968); this is a consequence of the topological division of space in 3 disjoint sets: interior, boundary, exterior (chapter 21).

This generalizes for higher dimensions. A test for a point in a line is a special case (no triangles!) but follows directly from the order of points on the line (x > a, x < b) (Figure 397)

11.2 Point in Simplex with codimension >0

If co-dimension is not 0, then one must first determine if the point is on the span of the simplex. Consider a point and a line (Figure 398). We determine first, if the point is on the flat AB or not, i.e. if ABX are collinear. For this determine the size of the area ABX and test against 0. If not 0 then point is not on in the span of the simplex AB.

This is the general test for a point to be in a flat or not (see chapter 20). If the given simplex of rank *n* and the point together determine a new simplex with rank *n+1*, which means *det (n,x) > 0*, then *X* is not inside.

If the point is in the flat spanned by the simplex then use the test for codimension 0 given above to determine if the point is inside the simplex or not.

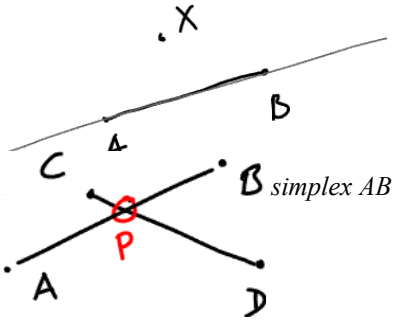


Figure 399: P is the intersection point of the two 1-simp

12. Intersection Point of Two *1*-Simplices

Two simplices intersect, if they have points in common. One could separate the test for intersection from the calculation of the intersection geometry. The general case of intersection of 2 *n*-simplices does not result in an *n*-simplex (Figure 383) and will be dealt with in the next chapter. In preparation, the special case of computing the intersection point of two *1*-simp is given here (Figure 399):

Figure 400: Three configuration where the 2-simp do not intersect.

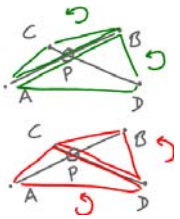


Figure 401: Four ccw tests to see if P is

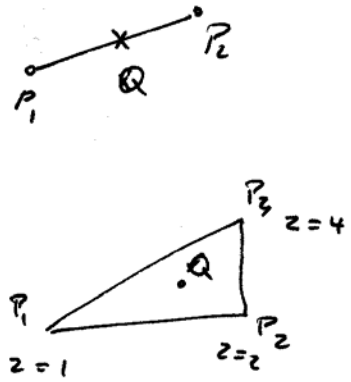


Figure 402: Interpolation problems

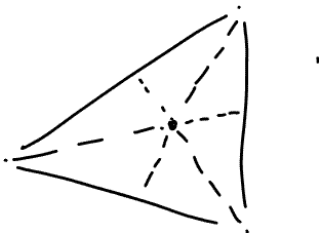


Figure 403: Barycentric coordinates

To determine the intersection point, the formula for intersection of flats is used (see chapter 20 and 21), but the computed intersection of the two infinite lines is not necessarily a point within the *1*-simplices (Figure 400). It must be inside the quadrilateral *ADBC*. This can be tested with the *ccw* predicate (Figure 401): the following four values must all be true: *ccw(ABC), ccw(ADB), ccw(ADC),* and *ccw(BCD).*

13. Interpolation and Contour Lines

An often encountered task is the interpolation of a point into a simplex with codimension 1 (i.e., a line in *2d* space or a triangle in *3d* space). Simplices are flat and permit linear interpolation.

Expressing the position of the new point as a parameterization, one can calculate the weighted average of the values for the boundary points. This can be generalized for linear interpolation in any dimension. It uses a barycentric coordinate system (Figure 403).

$$\lambda_1 \cdot x_1 + \lambda_2 = y \qquad \lambda_1 \cdot x_1 + \lambda_2 \cdot y_1 + \lambda_3 = z$$

$$A\,L = y \qquad A\,L = z$$

$$A = \begin{bmatrix} x_1 & 1 \\ x_2 & 1 \end{bmatrix} \qquad A = \begin{bmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{bmatrix}$$

$$L = [\lambda_1, \lambda_2] \qquad L = [\lambda_1, \lambda_2, \lambda_3]$$

$$y = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} \qquad z = \begin{bmatrix} z_1 \\ z_2 \\ z_3 \end{bmatrix}$$

$$L = A^{-1}\,y \qquad L = A^{-1}\,z$$

$$y_q = L \cdot [x_q \;\; 1] \qquad z_q = L \cdot [x_q \;\; y_q \;\; 1]$$

13.1 Contour Lines

To determine the pieces of a contour-line in a triangle is a useful function which can be used to construct contour lines for triangulated surfaces later. The endpoints of the contour lines result from the intersection of the boundary *1*-simp with the horizontal planes.

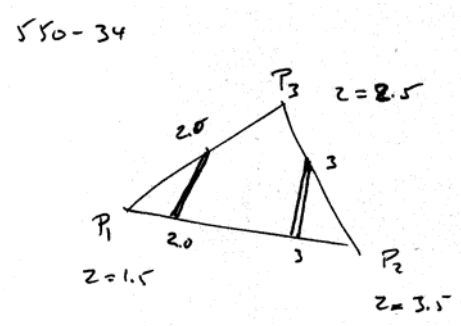


Figure 550-34

14. REPRESENTATION IN DATABASE

Simplices are represented in the database as entities. For each simplex, we can define a *relation* that gives the dimension of the simplex. Points are *0*-simplices.

The relation *boundary* gives to each simplex the boundary simplices. This gives for the join operation a different implementation than the join for flats; it creates a simplex and inserts the boundary points in the relation *boundary*.

15. CONCLUSIONS

Combinatorial topology considers topological relation of simplices and figures constructed from simplices; it counts distinct elements and is closer to implementation than the point set topology with infinite set we have used in the previous two chapters.

The operations on simplices use topological and metric properties of the simplices. This brings together the geometric operations on flats (part 3 and 6) with topology (part 7)..Often metric operations are at the base for topology. For example, the orientation of a *2*-simplex (a triangle) is determined by testing whether the determinant is positive or negative.

Summary of operations applicable to simplices in general:

- Rank, dimension, codimension, and corank,
- Equality, orientation and reverse,
- Boundary relation and the converse co-boundary,
- Intersection test.

Metric properties are:

- Length, area, volume, generalized to size,
- Point in simplex test,
- Intersection between two *1*-simplex,
- Parameterization, used for interpolation.

REVIEW QUESTIONS

- How to compute the height of a point inside a triangle?
- What operations on triangles are necessary to construct the contour lines for a triangulated surface?
- Why is the join of two points not commutative? What is the difference between *join (a,b)* and *join(b,a)*?
- What are the topological operations and relations that apply to simplices?

- Which operations on simplices require a metric space?

Chapter 24 SIMPLICIAL COMPLEX

The program of combinatorial topology is the transformation of topological problems into algebraic questions. The primary method is a form of counting(Henle 1994, 5); for example, Euler's polyhedron formula. $F - E + V = 1$.

The simplices introduced in the previous chapter are in this chapter combined to complexes, which are triangulations of space. Simplicial complexes are collections of simplices, such that for each simplex all its bounding simplices are also part of the complex. Simplicial complexes for *2d* are triangulations, but simplicial complexes exist for all dimensions and the treatment is mostly dimension independent.

Geometric figures will be represented as subcomplexes and mapped to chains of simplices. This approach is viable, because most topological properties are independent of the details of the triangulation; any triangulation will produce the same result(Henle 1994, 157).

Simplicial complexes and their subcomplexes are useful in a GIS because in a simplicial complex

- the intersection and union of two arbitrary subcomplexes is again a subcomplex, i.e., a total operation;
- for subcomplexes the topological relation defined by Egenhofer can be derived algebraically.

Program: Transform all geometric operations into operations on simplicial complexes

Simplicial complexes can be generalized to cell complexes and then include graphs and raster representations as special cases. This chapter shows how to deal with geometry in a GIS; it uses finite representations that can be implemented in today's computer systems and stresses topological invariants over the vagaries of approximation of metric operations with floating point numbers.

> Terminology: all complexes in this chapter will be understood as simplicial complexes.

1. INTRODUCTION

This chapter shows how to represent arbitrary geometric figures that are closed under union and intersection and for which topological relations can be computed easily.

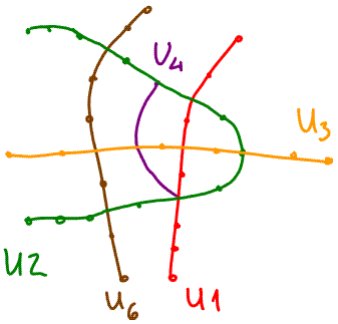


Figure 404: Metro lines in Vienna

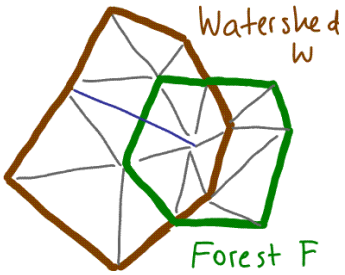


Figure 405: The intersection of a forest and a watershed

The geometric problem posed here is the one of union of figures, such that interesting geometric operations *op* distribute over union:

$op\ (a \cup b) = op'\ a \cup' op'\ b \quad where\ a \cap b = 0.$

Two examples: Compute the length of the metro network of Vienna, which is the union of the lines U1, U2, U3, U4 and U6 (Figure 404) or determine the area of a forest F within a watershed W (Figure 405). The lines and areas encountered in reality can be approximated with collections of simplices. Simplicial complexes are constructed such that they can be treated with algebraic methods.

Many discussions in GI Science centered around a representation for geometric figures and their topological relations(Dutton 1979). Rules restricting representations to subdivisions(Corbett 1975)) and later to (essentially) cell complexes (Frank 1983) were proposed. The use of the algebraic topology and specifically simplicial complexes were suggested first in 1986 {Frank, 1986 #325} and commercial implementation of the overlay operation using cell complexes appeared(Herring 1990; Herring 1991).

2. SIMPLICIAL COMPLEX

A triangulation in *2d*-space is a simplicial complex and we can generalize the notion to any dimension: a line or a graph is a *1*-dimensional complex, the triangulation of an area a *2*-dimensional one, and a *0*-dimensional complex is a collection of points.

Figure 406: A 2-, 1-, and 0-complex

A general simplicial complex is an arbitrary collection of simplices, resulting from the triangulation of some polyhedron(Alexandroff 1961). A simplicial complex is defined as a collection of simplices (Figure 406), such that

- for each simplex in the complex all the boundaries are also in the complex (i.e. all edges for the faces and all vertices for the edges) and
- the intersection of two simplices in the complex is empty or a simplex already in the complex.

A simplicial complex does not allow points which are not boundaries of an edge, edges that are not bounding two faces, etc. (Figure 407). The dimension of a complex is the maximum of the dimension of the simplices in the complex.

A simplicial complex of dim n consists of simplices, such that
- for each simplex all boundaries are in the complex,
- pairwise intersection is either empty or simplex (dim n-1).

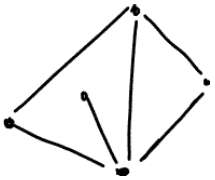


Figure 407: Not a complex!

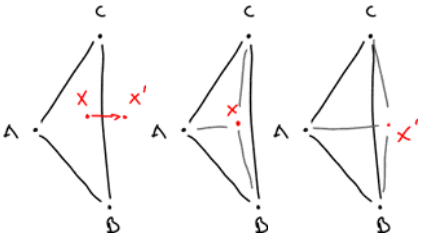


Figure 408: Points cannot move in a triangulation without violating explicitly stored relations

Terminology:
By complex we will in this chapter understand a subcomplex of a simplicial complex.

Operations with subcomplexes of a simplicial complex (e.g., a triangulation), are not requiring any metric operations. The representation of all geometric figures here will be by subcomplexes and all geometric operations in this representation will become algebraic computations (mostly operations with finite sets); the integration of geometries from different sources into a single complex is the operation, where the spatial information is converted into a discrete structure.

The simplicial complex is a most restricted and complete representation of the topological relations between the elements without reference to the coordinates. Changes in coordinates, for example by numerical operations which are only approximate, are not affecting the topology. This is the fulfillment of the motto: topology determines, metric refines(Frank and Kuhn 1986; Egenhofer and Sharma 1992). For example in Figure 408, the movement of X to X' changes the positive orientation of the triangle XBC; it becomes X'CB with a negative orientation. A representation using only coordinate values would require a test for X inside of ABC based on approximate metric properties. If the figure is represented as a complex, the topology is fixed.

The definition of a simplicial complex corresponds to what I initially called 'completeness of incidence' (Frank 1983; Frank and Studenmann 1983) and which Kuhn and I refined later, using combinatorial topology(Frank and Kuhn 1986). A simplicial complex lists explicitly a small number of relations to preserve the topological structure even in the presence of errors in the metric processing. Numeric problems cannot change the topology in a simplicial complex!

The simplicial complex represents all the intersection between the closure of the faces. This can be generalized to an abstract notion of complexes over arbitrary sets and their intersections. The complex then is the *nerve* of a system of sets(Alexandroff 1961, 39).

3. Directed Subcomplexes Represented as Chains

A complex *K* is directed, if every edge of *K* is given a direction from a start node to an end node, and every face a direction around the polygon(Henle 1994 185). A directed subcomplex is a subset of the oriented simplices of a complex, such that the subcomplex is a complex. A *k*-subcomplexes can be represented

by a sum of *k*-simplices, expressed as an *k*-chain. A simplicial *k*-complex (or a subcomplex) is a sum of *k*-simplices from the complex:

$C = a_1 S_1 + \ldots a_n S_n = \Sigma a_i S_i$

where the a_i are integers. $a_i = +1$ indicates that the simplex is in the chain, $a_i = -1$ that the reversed simplex is in the chain, $a_j = 0$ means that the simplex S_j is not in the chain. The empty chain is the chain with all factors 0. The addition of chains must be defined such that

$f(c_1 + c_2) = f c_1 + f c_2$.

The representation of the chain as a sum of products is similar to vectors, except that the factors are integers (and mostly restricted to *+1, 0,* or *-1*). We define a multiplication with *-1* in the usual sense and say that it reverses the complex by reversing all the simplices. For the addition, we will use four different rules:

OR: 1 + 1 = 1, 0 + 1 = 1 + 0 = 1, 0 + 0 = 0
XOR: 1 + 1 = 0, 0 + 1 = 1 + 0 = 1, 0 + 0 = 0 -- *generalized: addition mod 2*
AND: 1 + 1 = 1, 0 + 1 = 1 + 0 = 0, 0 + 0 = 0 -- *this is called an idemgroup (x=-x)*
Signed addition: 1 + 0 = 0 + 1 = 1, 0 + 0 = 0, 1 – 1 = 0, -1 + 0 = 0 – 1 = -1

With these rules for addition, four different kinds of addition of chains are executed pointwise, like vector addition or the addition of polynoms:

$c_1 = \Sigma a_j s_j \quad c_2 = \Sigma b_j s_j$
$c_1 + c_2 = \Sigma (a_j + b_j) s_j$ -- *where + may be any of the four operations.*

If the rule for additions has the properties of a group (XOR, and AND), the corresponding additions of chains are groups as well.

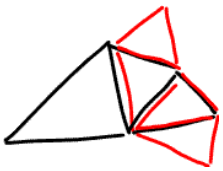


Figure 409: Union and difference of two chains

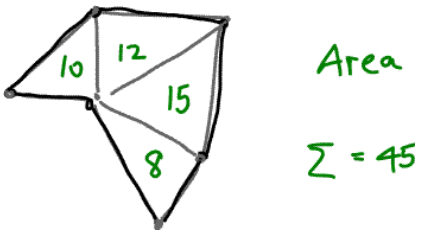


Figure 410

3.1 Unions and Intersection of Chains

The area of two complexes given as *2*-chains of the same simplex is the sum of the two chains, using the *signed addition* rule. The intersection of the chain is the result of the AND rule.

3.2 Size

Size Operations distribute over union and intersection, as desired. For example, the area of two disjoint *2*-complexes is the area of the union of the two complexes.

3.3 Boundary Operator for Chains

$\delta (a \cup b) = \delta a \cup \delta b$

The boundary operator for simplices gives as a result a chain: the boundary of a *1*-simplex (an edge) is a *0*-chain with two elements: the start (factor +1) and the end point (factor -1). In Figure 412, $\delta(a) = -B + C$. This boundary operator carries over to sums of chains, such that the boundary of a sum is the sum of

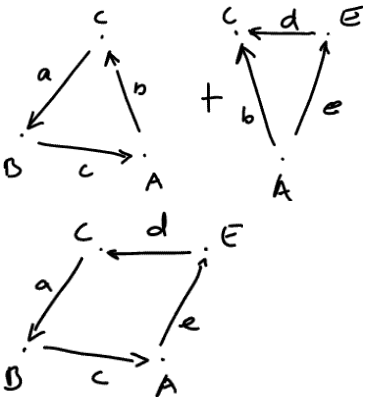


Figure 412: Boundary of sum is sum of boundaries (unoriented simplices)

boundary distributes over sum:
$\delta(\Sigma c_i) = \Sigma (\delta c_i)$

$\delta (\delta (r)) = 0$ *iff r a 2d-area*

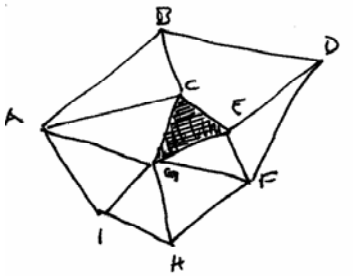


Figure 413: Subcomplex with hole

Figure 414:Skeleton, interior skeleton and interior

the boundaries, using the *signed addition* rule for the coefficients a_i:

$\Delta W = a + c + b$
$\delta V = e + d + b' = e + d - b$
$\delta (W+V) = a + c + e + d$

The boundary operation so defined works even for complex with holes (Figure 413). The result is a single chain, not separated in outer and inner boundaries. The boundary of the boundary of a 2-d area is 0; this can be used as a test.

Algebraic topology develops the theory of homology, which is a method to capture invariants of surfaces. It gives a justification why the approach suggested here works: Most topological properties of figures are independent of the triangulation selected (General invariance theorem,(Henle 1994, 157).

Let T be a triangulatable space of dimension two, that is, a complex composed of simplexes of dimension two or less. Then the homology groups of T are independent of the choice of triangulation.

I have not yet found a direct application of homology for GIS so far. It is useful for a generalization of the Euler polyeder formula to become the Euler characteristic of a surface and contributes to the solution of the 'map coloring problem', which says that all maps can be colored with just 4 colors.

3.4 SKELETON OF COMPLEX

The *k-1* skeleton of a *k*-simplicial sub-complex is the set union of all the *(k-1)* boundaries of the components (Figure 414). The skeleton of an area (i.e., a sub complex with triangles) is a set of boundary lines(Egenhofer 1989). It can be computed as the sum of the boundary of each *k*-simplex in the chain, using the OR rule for the addition

The *interior* skeleton (Figure 414) is defined as the skeleton minus the boundary (i.e., only the interior boundaries). The interior skeleton can be computed as the sum of the boundary chains, using the AND rule.

3.5 INTERIOR OF COMPLEX

The interior of a complex consists of all faces minus the boundary. If a *k*-complex is given by a set (or chain) of *k*-simplices, the interior consists of these faces, their inner skeleton and again the boundaries of these (Figure 414).

4. TOPOLOGICAL RELATIONS BETWEEN 2D SIMPLE REGIONS

Egenhofer has identified (see chapter 22) eight topological relations between simple regions(Egenhofer 1989), which are derived from the intersections of the interiors, the intersection of interior and boundary and the intersection of boundaries. The classification whether these intersections are empty or not determines the topological relations. To determine the topological relations between two regions given as simplicial subcomplex, we have to compute their boundaries and interior and then to test whether the intersections are empty. These operations are all algebraic.

The argument is given for two *2d* regions, each a subcomplex of a complex and given as a *k*-chain. The case of a *1d* line and a *2d* region is considered in the third subsection. The approach generalizes to higher dimensions, but Egenhofer's definition of relation is only given for the relations between two *2d* regions.

4.1 COMPUTE BOUNDARY AND INTERIOR

To compute the boundary and the interior of the two regions for testing Egenhofer relations must be attentive to the difference between the notion of boundary in point set topology (which Egenhofer's definition use) and the boundary operation in combinatorial topology, which is used for the implementation.

The *interior* of a *2*-subcomplex is the *2*-chain of triangles, plus the *1*-chain of the interior skeleton, plus the *0*-chain of the interior point.

The boundary of a *2*-subcomplex is the boundary operator applied to the *2*-chain, which gives a *1*-chain of the *1*-simplices in the boundary. Then determine the skeleton of this *1*-chain, which gives a *0*-chain, which are the points in the boundary (why is it not the $\delta(\delta(r))$?).

4.2 INTERSECTION TESTS

Simplices in a simplicial complex have only simplices in common which are part of the complex, there are no other intersections. Only simplices of the same dimension can intersect and if they are intersecting then they are equal—this converts the test for intersection in a test for equality!

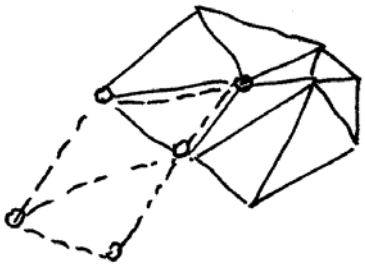


Figure 415: Intersection of interior skeleton only possible if the faces intersect

Figure 416 shows which tests must actually be done: The interior consists of a *2*-chain, a *1*-chain and a *0*-chain, the boundary consists of a *1*-chain and a *0*-chain. Only intersections of same dimension must be checked. We can exclude the intersection of the interior *1*-chain and the *0*-chain: two complexes can only have common interior skeletons, if they are common interior faces (Figure 415).

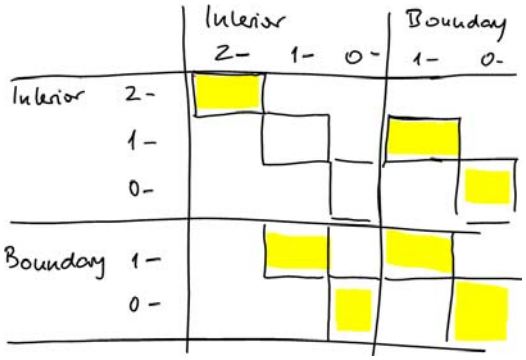


Figure 416: The seven intersections necessary

4.3 EGENHOFER RELATIONS BETWEEN 1D REGION AND 2D REGION

For the intersection between two *1*-complexes or a *1*-complex with a *2*-complex, the tests reduce further. For the *2*-complex, the interior faces can be ignored and we have to consider only the interior skeleton (*1*-chain and *0*-chain). For a *1*-complex, the interior is the interior skeleton (*1*-chain and *0*-chain) and the boundary is a *0*-chain.

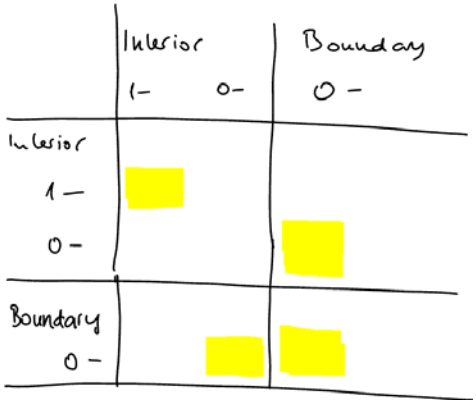


Figure 417: The intersections for the relations with a line

5. SUMMARY

Simplicial complexes and specifically subcomplexes are the representation for which geometric operations, especially sum and intersection, are total. The topological relations are the result of simple arithmetic and set operations and do not rely on metric operations with approximate floating point numbers. The next short chapter shows how two complexes are integrated to form a

single complex for which both are sub-complexes, such that the above described operations apply.

Simplicial complexes are triangulation and generalize to complexes built from cells. Most of the principles of simplicial complexes apply directly to cell complexes. A triangulation of a given situation includes more geometric objects than the same situation represented by cells, but operations on cell complexes require more complex implementations(Herring 1987; Herring 1990; Herring 1991). The theory is better explained with simplicial complexes, even if commercial GIS use cell complexes for performance reasons. Alternatively, better performance of simplicial complexes may be achieved with the reduction in the number of objects may also be achieved with the methods necessary for multiple representations (Lieblich and Arbib 1982; Minsky 1985; McKeown and Lai 1987; Beard 1988; Buttenfield and Delotto 1989; Günther 1989; Timpf, Volta et al. 1992; Buttenfield 1993; Frank and Timpf 1994; Sester 1996) and different levels of detail {Frank, 1986 #325}.

REVIEW QUESTIONS

- Why is the incremental overlay method less sensitive to the problems of approximate calculation with coordinates?
- Why is this called algebraic topology? What is it contrasted with?
- What is the difference between a cell complex and a simplicial complex?
- What is the definition of a simplicial complex?
- What is the boundary of a boundary? Where is it used?
- Why is Figure 407 not a simplicial complex?

Chapter 25 OPERATIONS FOR COMPLEXES

We have seen that the intersection of two triangles is not a single triangle, but 4 triangles and therefore intersection is not an operation applicable to triangles resulting in a triangle (figure in previous chapter 24).A general method to intersect any geometric figures can be given; it reduces to merging the two figures as two complexes to a single complex and then determine the intersection of the two subcomplexes. If we take each triangle as a simplicial complex then the intersection operation is just merging the two triangles, which are complexes, and determining the intersection of the two subcomplexs. This operation is closed; the result is a subcomplex of the complex and can be used for other operations!

This chapter describes the operations necessary to manage the simplicial complex and to integrate two complexes in a single one. Simplicial complexes are built according to a parsiomous principle(Knuth 1992, 62): no inconsistency can occur if nothing is ever tested for which the answer can be deduced from previous tests.

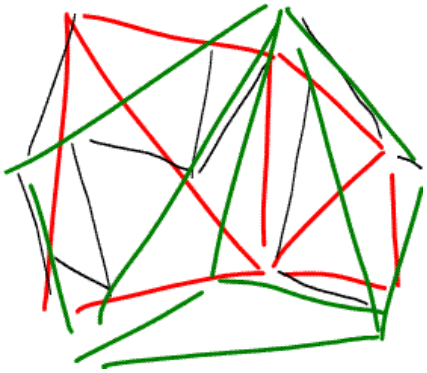


Figure 418: Example of the merge of two complexes

This merging of complexes achieves the geometric part of most GIS operations which is an overlay operation. For example the overlay of the subdivision of an area into parcels and a valuation map let us compute the value of each parcel (Figure 419, Figure 420). Assume that the two subdivisions are given as simplicial complexes; the overlay operation reduces to the integration of the two complexes into a single one, of which each is then a subcomplex. Then the intersection of the two subcomplexes is just an intersection two chains.

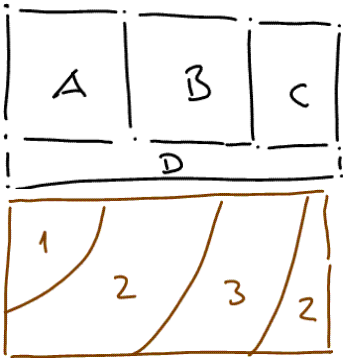
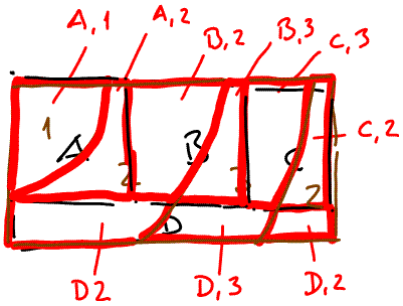


Figure 419 Parcels and valuation of a piece of land



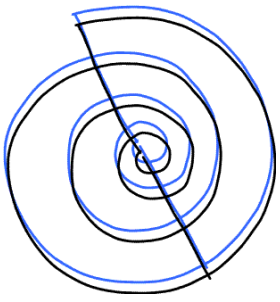


Figure 421 Test figure to check overlay operation

Figure 420: Overlay of land parcels and land value

1. Overlay Operation

The combination of two subdivisions is regularly occurring task in applications. It is called overlay operation. Examples are the overlay of the ownership partition and the valuation partition of some land (Figure 420), but the same operation is used to assess how much forest area falls into a town, computed as the overlay of land use with the political boundaries, etc. The operation consists of a geometric and a thematic part: in the geometric part, the smallest common areas are found and then the attributes of these faces are computed. In this chapter, only the geometric part is discussed.

The result of an overlay operation is a subdivision, where each cell is entirely included in a cell of each of the given subdivisions (Figure 420, with indications from which original cells a new cell derives); with the result of the overlay, the value of each of the original parcels can be computed as the area of one of the cells with uniform value times the value and then summed for the parcel. For example, the value of the parcel *B* is the area of *(B,2)* times value *2* and area of *(B,3)* times value *3*. The overlay operation is the common first step in all similar operations.

The overlay operation has been one of the oldest and most difficult problems for GIS.. Chrisman et al. has published a first approach as WHIRLPOOL(Dutton 1979). Properly working implementations were difficult to achieve; first, because inconsistency were introduced by the approximation of real numbers with computer arithmetic and, second, by the many special cases, in particular areas with holes. In the mid 1980s a U.S. Federal agency tested a number of commercially available overlay operations and all failed on some inputs (Figure 421 shows a difficult input, which creates many sliver polygons)!

We face these difficulties by two measures:

(1) Complexity and special cases are reduced by the restriction of all figures to simplices.

(2) Avoiding deducing a topological relation from coordinates twice reduces the influence of errors induced from approximate computations. This *parsimonious principle* (Knuth 1992, 62) was suggested in (Frank and Kuhn 1986) and by Steven Fortune(Knuth 1992). The result of a decision based on

coordinate calculations may contradict another approximate calculation using the same coordinates computed earlier. These differences in the calculation with approximations result—from the point of view of the logic of the algorithm—as inconsistencies in the data structure, which algorithms cannot tolerate and stop. Deriving the topological situation only once avoids this difficulty. An alternative approach is using computation with realms(Schneider 1997).

Parsimonious principle: "do not ask dumb question", i.e. questions that can be answered from what is already known.

2. MERGING COMPLEXES

Merging two complexes is approached as the element-wise insertion of the simplices of the smaller one into the larger one (Figure 422). Specific operations which insert points, lines and faces into a complex are required, as well as an operation to create a new, empty complex. With these operations, any complex can be constructed as merging two complexes.

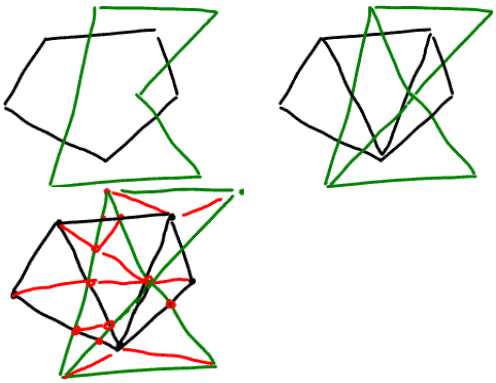


Figure 422: 2 regions, triangulation of the two regions, integration

The discussion here is in terms of operations for a *2*-complex; the extension to higher dimensions is left for future extension.

3. STARTING CASE: CREATE EMPTY |COMPLEX

The starting case is an empty complex. To avoid difficulties with the outer edge, we start with a *2*-dimensional simplicial complex which triangulates the sphere (Figure 423). Note that only the 2-simp A and B should be used and the other 'triangles" are only to complete the figure and assure that all nodes have the same structure. It also achieves that the computations are restricted to the part of the projective plane which has a consistent orientation.

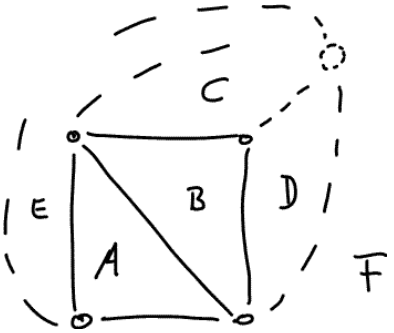


Figure 423: Triangulation of the sphere with 6 2-simp

4. TEST FOR POSITION OF POINT IN A COMPLEX

The first step in the insertion of simplices in a complex is the determination of the affected simplices. This reduces primarily to the determination of the position of a new point within the complex. A point can be

(a) Coincident with a *0*-simp already in the complex
(b) Incident with a *1*-simp (line segment) in the complex
(c) Incident with a *2*-simp (triangle) in the complex

No other case can occur, because the triangulation covers the whole projective plane.

The test uses the *CCW* predicate. In principle, one triangle after the other is tested with the point, but this is not parsimonious, the same questions are asked repeatedly, once for inclusion in the triangle on the left side of a line, once for inclusion in the triangle on the right side of the line. Using the CCW predicate to determine if the point is left of a line in the following algorithm has been suggested by Stolfi and Guibas(1982):

```
Start with an arbitrary 1-simp.
If point is left of 1-simp then
  Select next left 1-simp at end and test point
         against this one
     If point is left then
         Select 1-simp at end and test point againt
                  this one
         If point is left then point is inside the
                  triangle found
     else
         select 1-simp right at end and test point
against this one …

repeat till point was left to the past three lines.
```

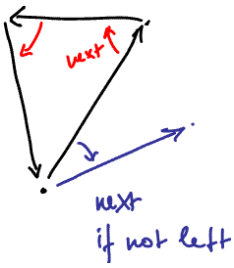


Figure 424: Repeated CCW test

The idea is to restrict the search to the subspace formed by the infinite extensions of the *1*-simp where we have not yet tested that the point is outside (Figure 425). If the test with the current line is true then continue with the next *1*-simp of the triangle (in positive turning direction), if not continue with the next *1*-simp of the triangle on the other side.

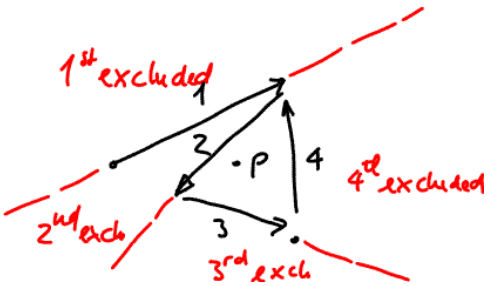


Figure 425: Each test excludes a half-space

If three sides in a row test positively, then the point is inside.

Figure 426: Split 1-simp

The algorithm given by Guibas and Stolfi does not deal only with points in general position. An implementation for GIS must detect when the point is incident with a line or coincident with a point in the complex first.

5. Adding a Point to a Complex

Adding a single point maintaining the complex structures is the building block of the algorithm. Adding a point in a complex must make this *0*-simp a boundary of at least one *1*-simp, and this *1*-simp must be a boundary of some *2*-simplex. Three cases are differentiated, depending where the point is lying.

Case a: New point coincident with other point—nothing needs to be done for the geometry

Figure 427: Barycentric split of 2-simp

Case b: New point incident with line. The *1*-simp must be split in two *1*-simp and the adjacent triangles split as well (Figure 426).

Case c: New point incident with triangle: The triangle must be split in three and three new *1*-simp introduced (Figure 427).

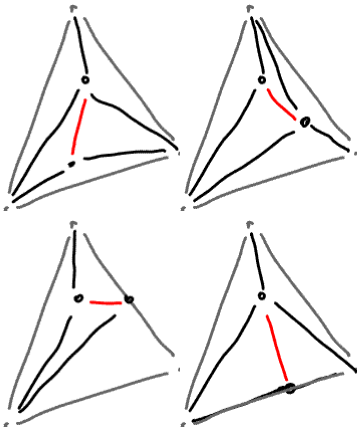


Figure 428: Different cases on insertion of 1-simp where inserting the points inserts automatically the line

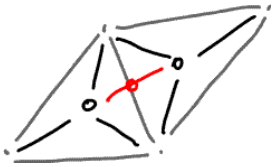


Figure 429: Endpoints of line in neighboring triangles

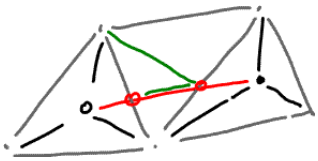


Figure 430: Endpoints not in neighboring triangles

6. Adding a Line to a Complex

Adding a line of a complex requires first to add the two *0*-simp that are the boundary of the line. Assume this is done and we have only to add the 1-simp between two 0-simp of the complex.

Case 1: Both boundary points are within the same triangle (i.e., the 1-simp is completely in one triangle) and nothing additional need be done; with the insertion of the two boundary points the line is already inserted (this is even the case when the second point is coincident with one of the newly inserted 1-simps) (Figure 428).

Case 2: The boundary points are in two different triangles.

Subcase 2a: the points are in neighboring triangles: Insert the two points and determine the intersection point of the new *1*-simp with the boundary lines of the triangle. Insert this point and the new *1*-simp is also in the complex (Figure 429).

Subcase 2b: the endpoints are not in adjacent triangles. Insert the two endpoints and determine the intersection points with the 1-simp in the complex (not all need to be tested!). Insert these points and the 1-simp is also in the complex (Figure 430).

Note that the determination of the intersection points of *1*-simp is restricted to the boundary of the triangle in which one of the endpoint is lying, respective where a new point of the line is inserted. An efficient algorithm is proceeding from the start to the end of the new *1*-simp, inserting the start point, determining the intersection with the boundary and inserting this point, selecting it as the new start point of the reminder of the line to insert (Figure 431). This is using subcase 2b repeatedly till eventually subcase 2a applies, which then reduces to case 1. This shows that this approach can handle all insertions of 1-simplices.

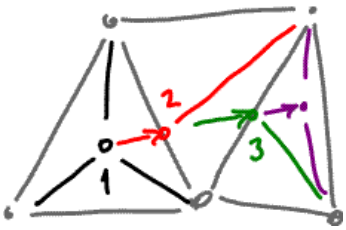


Figure 431: Gradual insertion of 1-simp

7. INTERSECTION OF SIMPLICIAL COMPLEXES

Intersecting two complexes means first to integrate the two complexes into one and represent the two given complexes as subcomplexes of the merged complex, and then to compute the intersection as an operation on integral chains, representing the two subcomplexes.

Merging the two complexes is done with adding one of the two complexes element by element to the second one complex. The two complexes *A* and *B* are given (Figure 432). Take the one complex *A* and determine the skeleton of the simplex *B*, which is a *0*-chain and a *1*-chain. The simplices of these chains are now added elementwise:

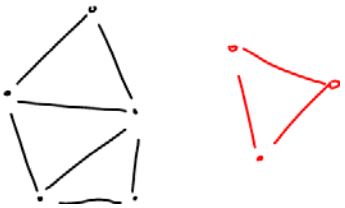


Figure 432 Two complexes A and B

Step 1: Start with the integration of the points from the *0*-chain. Repeat for every point: The point is (Figure 433)

- coincident with a point of *A*, nothing needs to be done
- incident with a *1*-simp of *A*, split the line and both neighboring triangles
- inside a *2*-simp of *A*, then split the triangle

Details of these operations have been discussed before (section 4); the result is shown in Figure 433.

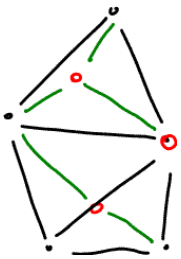


Figure 433: The three cases for integration of points

Step 2: integrate the *1*-simps of *B*. Repeat for every *1*-simp in *B*: A *1*-simp of *B* can

- coincident with a line in *A'*: nothing needs to be done.
  or
- intersect one of the *1*-simp of the augmented complex *A'* (i.e., *A* plus all the *0*-simps from *B*). Compute the intersection points with *1*-simp in *A'* and insert these points in *A'*. This also inserts the line

For the example, one line needs to be added, which intersects another line; it is necessary to insert a new 0-simp, which then inserts automatically also the two *1*-simps (Figure 434).

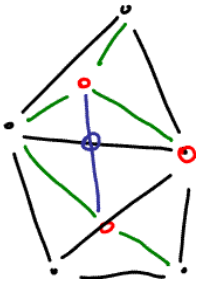


Figure 434: Adding one more 0-simp and two 1-simp

With this a complex integrating both *A* and *B* is established and *A* and *B* can both be expresses as a chain of *2*-simp. The intersection is then the intersection of the two chains and one could also determine Egenhofer relations as computations with the chains.

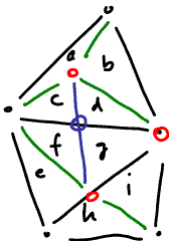


Figure 435: Labeled 2-simps

A = a+b+c+d+e+f+g+h+i

B = d+g

A ∩ B = d+g

8. MAINTAIN SPLITTING HISTORY OF A LINE

The approximation of coordinates by floating point number introduces small errors. If a line from *A* to *B* is repeatedly split, the splitting points P_1, P_2, .. P_n are not exactly situated on the original line (Figure 436). If one is later taking the azimuth of any of the resulting segments P_i to P_{i+1} the value obtained may be considerably off.

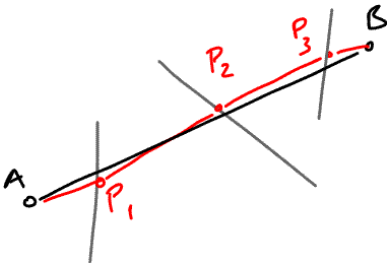


Figure 436: A line with multiple splitting points

In each case of a split, remember the original and how it is split. This gives a relation: *consist_of*. The GIS must maintain the original line *AB* and record that it is split into lines AP_1, $P_1 P_2$, etc. Similarly for areas: The two regions representing the object geometry are given—after triangulation of the region—as *2*-chains. After the integration we must still be able to reconstruct the original triangulations.

In Figure 437 the two triangles *A* and *B* are integrated. The original situation is given by two boundary relations: *0*-boundary, which connects a *1*-simp with the *0*-simps, and the *1*-boundary relation, which gives for each *2*-simp the bounding 1-simps. The Figure 438 gives the original boundaries plus the addition of new boundaries as they are split (but no existing entries in the relations are dropped!). In addition, history relations are maintained, *1*-history gives for the new *1*-simps that other *1*-simp they are part of, and similar for the *2*-history which gives the parts for the original two triangles *A* and *B* (Figure 439).

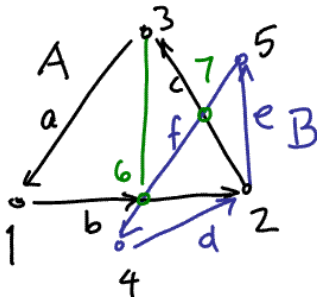


Figure 437 Two triangles A and B to integrate

δ_0

a	1, -3
b	-1, 2
c	-2, 3
d	-4, 2
e	-2, 5
f	-5, 4
g	6, -3
f_1	-5, 7
f_2	-7, 6
f_3	-6, 4
c_1	-2, 7
	-7, 3

δ_1

A	a b c
B	d e f
A_1	$a\ b_1\ g$
A_2	$b_2\ c\ -g$
A_3	$-g\ -f_2\ -c_2$
A_4	$b_2\ c_1\ -f_2$
B_1	$f_3\ d\ -b_2$
B_2	$b_2\ c_1\ f_2$
B_3	$-c_1\ e\ f_1$

Figure 438: The integration of A and B

1-history

b	$b_1\ b_2$
c	$c_1\ c_2$
f	$f_1\ f_2\ f_3$

2-history

A	$A_1\ A_2$
A_2	$A_3\ A_4$
B	$B_1\ B_2\ B_3$

Figure 439: The history relations

9. SUMMARY

Simplicial complexes and specifically subcomplexes were the missing concepts to compute union and intersections of arbitrary figures. If geometric figures are represented as subcomplexes of a single complex then the operations necessary to compute intersection, unions or the topological relations based on the 4 or 9 intersection become operations on chains. .

The integration of two simplicial complexes in 2 dimensions needs two steps, which are repeated:

- Integrate a point from one simplicial complex into the existing one. This has been covered as the insertion of a point into a triangulation.
- Integrate a line from one simplicial complex into the existing one. This is covered in the first subsection, differentiating several cases.

An additional operation to integrate the faces is not required (for 2-dimensional complexes). However, relations that record how simplices are subdivided during the integrations steps are necessary to have access to the original definition of the figures and avoid influences of the accumulation of approximation errors when computing intersection coordinates.

PART NINE AGGREGATES OF LINES GIVE GRAPHS

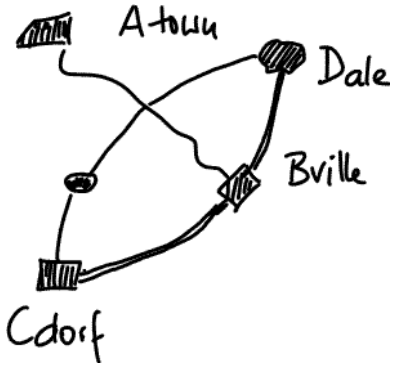


Figure 440Towns and streets between them

Figure 441Trade flows between European countries

This part discusses special simplicial complexes, namely 1-complexes, complexes that consist only of lines. Connections between points are often used to conceptualize our world: roads between villages (Figure 440), telephone lines between buildings, but also rivers flowing towards lakes and the sea. Such networks are seen in all applications of GIS: concrete representation of linear features like street networks, rivers, but networks are also used to conceptualize abstract situations like migration flows, trade between countries (Figure 441), etc. Networks can be shown well graphically. Maps show networks as lines and our concept of geography is influenced by the representation of space and objects in space as maps. The graphs discussed in this part are abstractions from lines on maps.

The abstract notion of network and the analysis of network properties are generally useful. Kirchhoff's laws, originally formulated for electric networks, apply in many similar situations. They state (1) that the sum of flows in and out of a node must be zero and that (2) that the sum of the differences in potential around a closed circuit must be zero as well.

This parts introduces properties of networks, abstracted to graphs of nodes and edges that have clean definitions and allow meaningful analysis operations. The foundation provided by graph theory contributes to the advancement of the topical sciences like regional economy, hydrology, transportation etc.

The first chapter introduces graphs as an abstract concept of nodes and edges connecting them. It introduces a number of notions that are widely applicable and ends with the description of an algorithm to find the shortest path in a network. The properties discussed in this chapter are invariant under topological and even more general transformations.

The second chapter then brings back the geometric aspects; it concentrates on graphs that are embedded in *2d* space and

shows how this additional knowledge can be used to improve the algorithm.

Chapter 26 ABSTRACT NETWORKS: GRAPHS

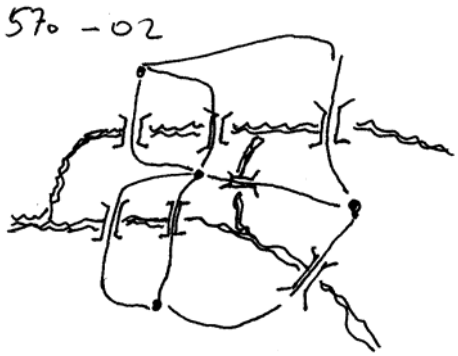


Figure 442: Königsberg and its islands

The famous mathematician Euler (1707 – 1783) regularly walked through the city of Königsberg and asked himself, whether he can have an evening walk, such that he crosses each bridge exactly once and return home (Figure 442)? The question identifies what is important in the situation and what can be left out: it does not depend on the form and position of bridges in Königsberg, but only on islands and the connections between them. This abstraction and the theory that belongs to it is today called *graph theory*. The simplification to nodes and connections between them yields a an elegant theory that answers many questions that depend only on lines and how they connect nodes, not on the particulars of the form of the lines or the position of the points in space.

Graph theory is motivated by spatial situations, but it is an abstract theory. Street networks, rivers and similar structures show properties, which are captured with the structure of graphs. They are all forms of connected lines and one can ask questions like: Is there a connection between *A* and *B*? What is the shortest path from *A* to *B*?

Graph—a bi-partite structure of nodes and edges, with an incidence relation.

Terminology:
An edge is incident with a node.
Two nodes are adjacent if connected by an edge.

A graph consists of nodes and the lines between them, which we call edges. Graph theory is constructed as a bi-partite algebra over nodes and edges and a relation incidence of a node with an edge. Graphs are invariant under topological transformation and other transformations that preserve the incidence relations. The embedding of a graph, which is necessary to show a graph, is arbitrary (Figure 443).

1. INTRODUCTION

Graph theory captures the connectivity in a situation. Street networks are an example, where a road connects towns, but airline connections form graphs as well as the water lines or the sewage pipes in a city. They are all forms of connected lines and one can ask questions like: Is there a connection between A and B? What are the neighbors of B?

Graph theory concentrates on the structure of connections between points. From a graph point of view, the map of Königsberg reduces to nodes and the connections, with arbitrary position in space. The graph in Figure 444 is equivalent to the original map (Figure 442). Graph theory is the geometry in which *incidence* and *adjacency* between points are the invariants; all other aspects can change without affecting the results. Even in this abstract form, the core of the problem is present and one can demonstrate why Euler cannot walk once across all bridges and reach home. Can you see the reason? Can you express it as an abstract rule?

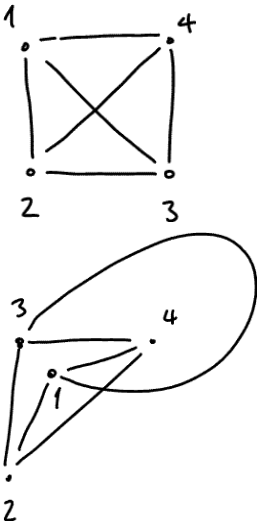


Figure 443: The same graphs with different embedding in 2dspace

Graph theory defines terms like path, walk, etc. in a strict way. Interesting are optimal paths, called shortest path between two nodes. Dijkstra has published in the early days of computers an elegant, non-trivial algorithm to find the shortest path in a graph(Dijkstra 1959).

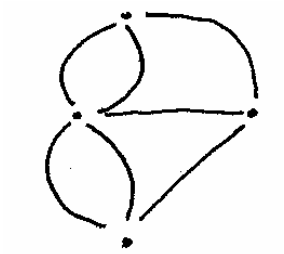


Figure 444: The graph theoretical essence of the map of Königsberg

2. ALGEBRA OF INCIDENCE, ADJACENCY, AND CONNECTIVITY

Graphs are bi-partite algebraic structures, which consist of *Nodes* and *Edges* and an *incidence* relation between them. The intuition for nodes is points (*0*-simplices) and for edges are line segments (*1*-simplices or *1*-cells). A graph is an abstract simplicial *1*-complex.

node = 0-simp
edge = 1-simp

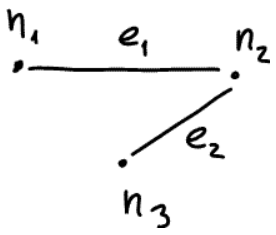
The function from edge to nodes is the boundary operation for a 1-simplex

Graph theory was developed early and the terminology in is often at odds with current terminology in other fields of mathematics.

2.1 DEFINITION

A graph consists of a set of Nodes $N = \{n_1 \ldots n_n\}$ and a set of edges $E = \{e_1 .. e_e\}$. The edges are not oriented and the edge $e_{ik} = (n_i, n_k)$ and the edge $e_{ki} = (n_k, n_i)$ are equivalent. The graph is a function from the edge to a pair of nodes (under the equivalence *Eq*: $(n_k,n_i)=(n_i,n_k)$) :

Figure 445: Graph (N, E, g) with $N = \{n_1, n_2, n_3\}$, $E= \{e_1, e_2\}$, $g(e_1) = (n_1, n_2)$, $g(e_2) = (n_2, n_3)$



g:: e -> (n,n)/Eq.

The triple (*N, E, g*) is a graph (Figure 445). The function *g* gives the incidence relation, from which an adjacency relation can be derived.

- **Incidence**: a node is incident with an edge if the edge starts or ends in it:
  incident (n,e) $\Leftrightarrow$ *g (e) = (n_1, n_2) && n==n_1 || n== n_2*
- **Adjacency** between two nodes means that the nodes are connected by an edge:
  adjacent (n_1, n_2) == exist e | incident (n_1, e) && incident (n_2, e)

2.2 WALK AND PATH

Two relations:
Incidence of node and edge
Adjacency of two nodes connected by edge

A *walk* is defined as a sequence of edges e_i such that e_i and e_{i+1} are both incident with the same node. A walk can contain an edge more than once (e.g., *{e_1, e_2, e_3, e_4, e_5, e_2}* is a proper walk, containing e_2 twice) (Figure 446). A walk between n_1 and n_m is defined as an alternating sequence of nodes and edges

n_1, e_1, n_2, e_2, ... n_j, e_j, $n_{(j+1)}$.... n_m
where for all i incident (n_i e_i) and incident (e_i, $n_{(i+1)}$)

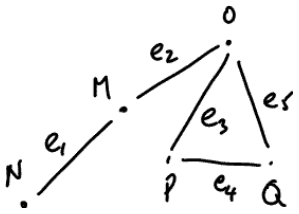


Figure 446: A walk

A walk is *closed* if the last edge e_j and the first edge e_1 are incident with the same node (Figure 447). A *path* is a walk such that no edge appears twice; a path can be closed and is then called a cycle (Figure 447). The *length* of a path is the number of edges it contains.

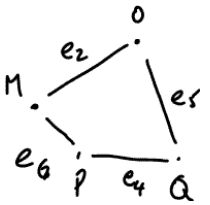


Figure 447: A closed path, which is a cycle

A path is called Hamiltonian, if it uses each node exactly once. A closed walk is Eulerian, if it uses each edge exactly once.

2.3 DEGREE OF NODE

The degree of a node counts how many edges are incident with this node (Figure 448).

2.4 CONNECTIVITY

A number of notions relate to the connectivity in a graph: Two nodes are connected if there is a path between them. All nodes that are adjacent to a given node are connected, but also all nodes that are adjacent to the connected nodes are indirectly connected. Adjacency is in this context called *directly connected.* The relation connection is transitive:

con a b and con b c => con a c.

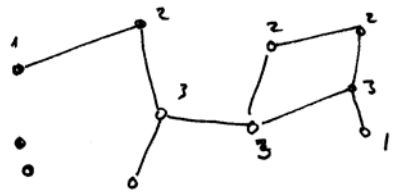


Figure 448: Graph with nodes labeled with degree

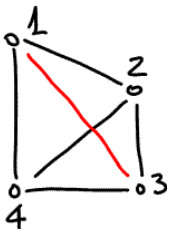


Figure 449: Completely connected graph

A graph, in which between any pair of nodes an edge exists, is called '*completely connected*' (Figure 449). A bridge is an edge, which if removed, disconnects the graph in two components.

The connectivity of a network can be measured by comparing the existing number of edges with the maximum number of edges possible. This connectivity measure can vary from *0* to *1*, *0* being a graph with no connectivity, no edges at all; *1* is obtained for a completely connected graph (Figure 449 above). A connected graph has minimum connectivity, if all nodes are connected to some other node, with a connectivity value of 2/m(Abler, Adams et al. 1971, 259).

connectivity measure

$$C = \frac{m}{\frac{1}{2}(n^2 - n)}$$

m = number of edges

n = number of nodes

minimum connectivity

$$C_{min} = \frac{n-1}{\frac{1}{2}(n^2-n)} = \frac{n-1}{\frac{1}{2}n(n-1)} = \frac{2}{m}$$

2.5 COMPONENTS OF A GRAPH

The components of a graph contain each all nodes which are connected (Figure 450). Connectivity is a transitive relation and forms equivalence classes which are the components. Components are the fixed points or closures of the connected relation; they contain all nodes that are directly or indirectly connected to a given start node.

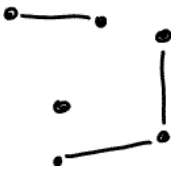


Figure 450: A graph with three components

To identify the components of a graph requires an inspection of every element of the graph. One can imagine, connectivity spreading out from a start node, first connecting the nodes with path of length 1 (i.e., directly connected), then connecting nodes with path of length 2, then with path of length 3, etc. (Figure 451)

We will in later sections see methods to maintain a graph connected (part 10).

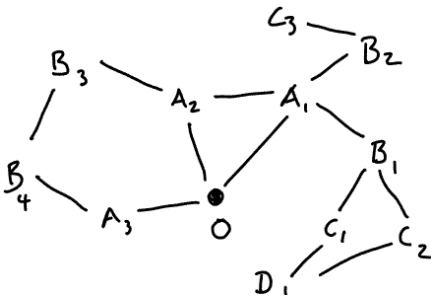


Figure 451: A nodes are directly connected to O, B nodes are connected by path of length 2, etc.

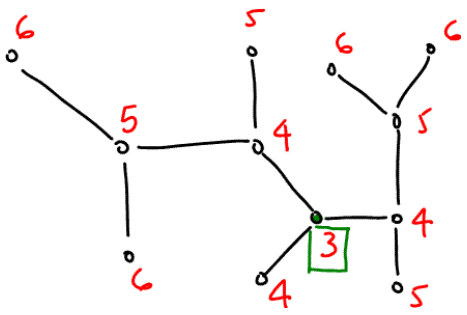


Figure 452: Radius of a graph: The nodes are marked with the excentricity

2.6 CENTER AND RADIUS OF A GRAPH

In analogy to the diameter and radius of a circle we can speak of diameter and radius of a graph, taking each edge as unit length. We start with defining the excentricity of a node: it is the largest distance to any other node in the graph. The radius of a graph is the minimum excentricity of any node; the diameter is the maximum excentricity of any node.

3. SPECIAL TYPES OF GRAPHS

A simple graph has only an incidence relation: does an edge start- or end at a given node. Edges can carry more information:

3.1 LABELED GRAPHS

In labeled graphs every edge has a label, which contains some information. Labels are functions from an edge to a value (Figure 453):

```
label: edge -> value
```

Labels on the edges are used in GIS to describe properties of the edges of a graph—width of a road, length of a road segments, or the cost of traversing the edge.

In a labeled graph properties based on the labels can be determined; for example the sum of the labels or the maximum or minimum label ("a chain is only as strong as its weakest link"). The sum of the length labels of a path is the length of the path.

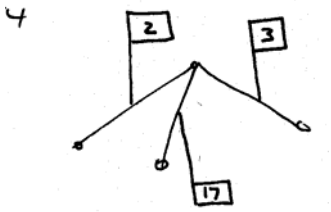


Figure 453: Graph with labels

A special case of a labeled graph is a **weighted graph**, where the labels are all positive numbers. For example a graph with labels indicating the length of the edge, is a weighted graph.

3.1.1 Directed graphs

In directed graphs, the edge (N_5, N_6) is different from the edge (N_6, N_5). Two nodes *A* and B are only adjacent if the edge *(A,B)* is in the graph; it is not relevant if an edge *(B,A)* is in the graph. Sometimes we speak of an oriented graph, which is a graph where each edge is given a direction and labels then reference this direction (e.g. flow graphs).

Directed graphs are used to model street networks, where some roads are restricted to 'one way streets'. One can either represent a one-way street by a directed edge and use two directed edges to represent the two directional lanes of a two way street (Figure 454). Alternatively, a oriented graph can be

labeled with labels ‘oneWay’, ‘oneWay the other direction’, ‘two ways’. In a graph where edges are labeled with their directed flow, Kirchhoff's law says that the sum of the flows in and out of each node is zero, which translates to: for each node, the sum of the labels on the adjacent edges is zero.

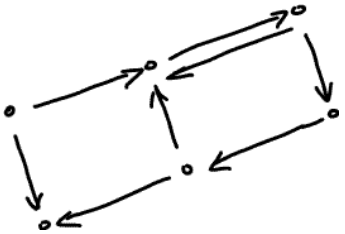


Figure 454: Two-way edges are modeled with two (anti-) parallel edges.

For directed graphs, the in-degree (edges ending at the node) and the out-degree (edges starting at the node) are differentiated.

```
indeg, outdeg :: n -> g e n -> Int
```

In a directed graph, a node *A* can be connected to *B* but *B* not connected to *A* (this is avoided in street networks!). For directed graphs, two forms of connectedness can be differentiated: A directed graph is strongly connected, if for any two nodes *A* and *B A* is connected to *B* and *B* is connected to *A*. If only one of the two connections exist, the graph is said to be weakly connected.

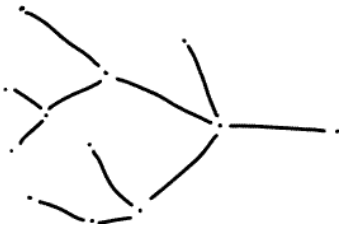


Figure 455: An acyclic graph

3.2 TREES

Certain applications lead to graphs that do not have cycles—for example, river networks (Figure 455); in such a tree, any two nodes are connected by exactly one path. If the graph has multiple components and any two nodes are connected by at most one path, then we have a forest (Figure 456).

Trees are often used to classify entities; taxonomy is a classification of terms and has typically tree structure. The hierarchical structure of political subdivisions form a tree (Figure 457) as do depiction of a person's ancestor (a family tree).

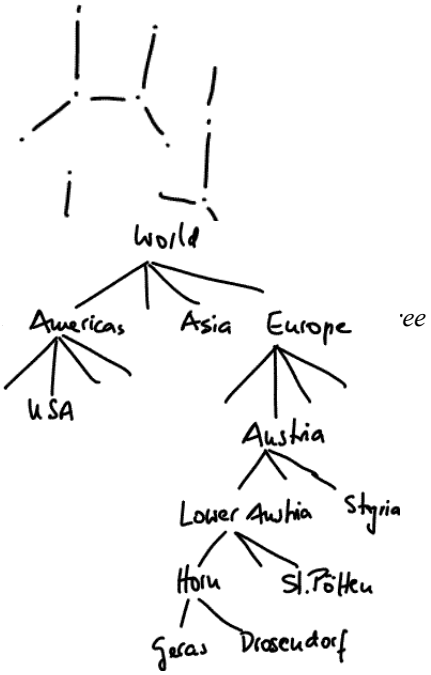


Figure 457: The tree of the political subdivision

Trees do have special properties that are useful when constructing algorithms(Samet 1989). If data is properly organized in a tree, one can search for an element by binary decisions between the left and the right subtree. Processing of a tree is a recursive procedure, which follows from the recursive definition of the data structure (see chapter 4). Decision can be analyzed using trees, where each bifurcation in the tree represents a decision; if observed values are given, then a decision tree can be reconstructed; this is a popular method in data mining [sheckar book].

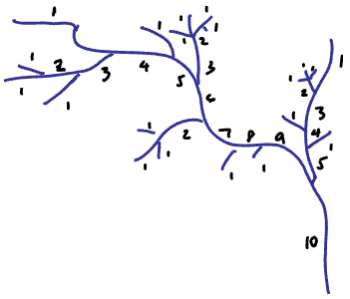


Figure 458: Stream order labels

In a directed tree, we can label edges from the roots forward, adding one for each successive edge and taking the maximum value at each node. For stream networks, these labels are called stream order (Figure 458). Real streams, but also other applications lead to graphs which do not have cycles following the direction of the edges (Figure 459). For acyclic graphs a partial order relation obtains and some acyclic graphs are lattices.

Figure 459: An acyclic graph

3.3 Bi-Partite Graph

A graph where the nodes can be separated in two sets such that no edge is between two nodes from the same set is called bi-partite. They are used to describe matching problems, for example matching people with a set of jobs. Petri-nets are another application for bi-partite sets; Petri-nets contain place nodes and transition nodes (Figure 460) and the nodes are marked by tokens, which move in time. A transition "fires" when both input nodes are marked by a token and then all the output nodes get marked.

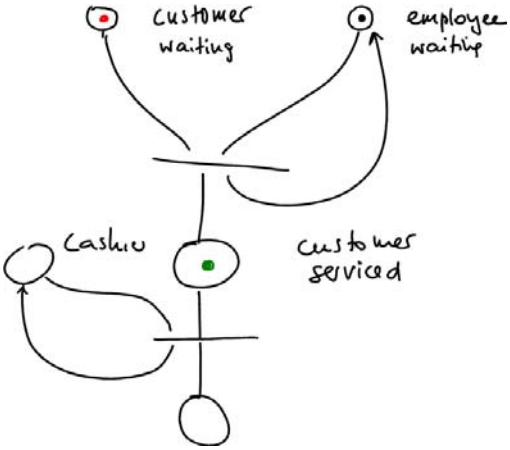


Figure 460Petri-net showing service to customer

3.4 Special Cases

Some special cases which are either potentially difficult to handle in an algorithm or cannot be represented are often excluded:

3.4.1 Loops

An edge that connects to the same node is called a loop (Figure 461).

$g(e_4) = (N_3, N_3)$

For most applications loops do not make sense, and algorithms assume that a graph does not contain a loop and fail if they encounter one.

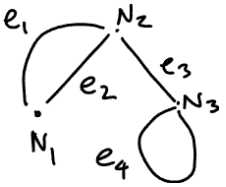


Figure 461: A graph with a multi-edge and a loop.

3.4.2 Multi-Edge (Zweieck)

If two edges run between the same two nodes, we say they form a multi-edge ('zweieck', Figure 461). Such edges are excluded in the definition of a graph where edges are identified by the pair of nodes they are incident with. If multi-edges are required for the application, then edges must have independent identifiers and we cannot just use the pair of node identifiers as identifiers for the edge.

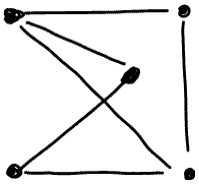


Figure 462: Planar graph: can be drawn in 2d without intersection of edges.

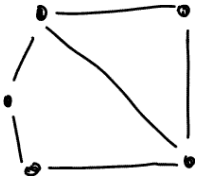


Figure 463: the same graph redrawn without crossing edges

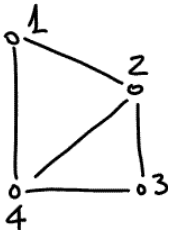


Figure 464: A simple graph

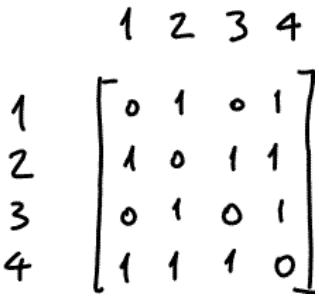


Figure 465: The adjacency matrix for the graph Figure 464

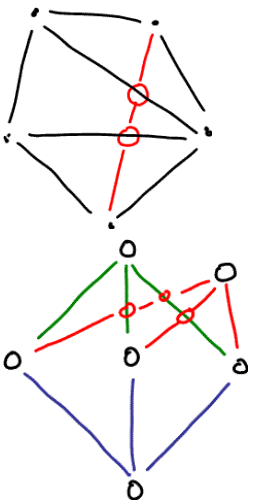


Figure 466: The two non-planar graphs K_5 and $K_{3,3}$

4. PLANARITY

A graph is said to be planar, if it can be drawn in *2d* space (a plane) such that no two edges cross—independent of location of nodes and form of edges.

The interesting question, whether a graph can be drawn in a plane without crossing of edges (other than those incident in nodes), can be answered without reference to the location of the nodes. There are two non-planar graphs (Figure 466), and any graph that contains one of the two cannot be drawn in a plane and all other graphs are planar.

5. REPRESENTATIONS

There are many ways to represent a graph. All are based on the storage of the incidence or the adjacency relation.

5.1 REPRESENTATION AS ADJACENCY MATRIX

A graph is represented by the adjacency relation: which nodes are incident with an edge. This can be captured, for example, in a square matrix, where for each combination of nodes 0 signifies that the nodes are not adjacent and 1 when they are adjacent.

The *adjacency matrix A(G)* of the graph *G* is an *n x n*-matrix where *n* denotes the number of vertices in *G*. Its entries a_{ij} are *1* if the (directed) edge from node *i* to *j* exist in the graph G and *0* otherwise. For non-directed graphs, the matrix is symmetric. The row or column sum gives the degree of a node, for directed graphs, the in- and the out-degree separately. The sum of all 1s in the adjacency matrix gives the number of directed edges (or twice the number of non-directed edges).

The adjacency matrix can be can be read as base for a vector space, even without coordinate values assigned to the points. This opens opportunities to apply methods from linear algebra to the analysis of graphs in the abstract [ref?].

5.2 CONNECTIVITY

The multiplication of an adjacency matrix with itself gives the connectivity of path length 2, further multiplication path length 3, etc. Figure 467 gives the square of the adjacency matrix of the simple graph from Figure 464; it shows the number of path of length 2 in this graph. Note that it contains in the diagonal the degree of the nodes; each edge starting at a node gives rise to a path of length 2 back to this node!

We can define a power function,

$A^2 = A * A$

$A^n = A * (A^{n-1})$;

The fixed point ($A^n = A^{(n+1)}$) gives the longest path (length = n) in the graph. The sum of all connectivity matrices ($A + A^1 + A^2 .. A^n$) gives the total connectivity. The entries of the k-th power A^k of $A(G)$ count the walks of length k (i.e., with exactly k edges) between a fixed starting point and the endpoint in G.

The matrix $I - A$ is invertible iff the graph does not contain any directed cycles. The inverse $(I-A)^{-1}$ gives the number of directed path between two nodes (because $(I-A)^{-1} = I + A + A^2 + A^3 ...$). Spectral theory of graphs gives more connections between linear algebra and graph theory.

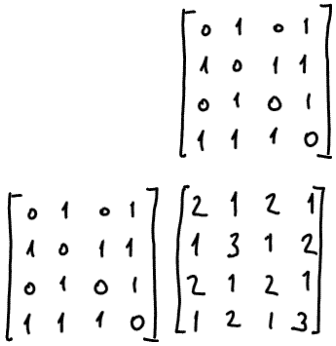


Figure 467: The square of the adjacency matrix from Figure 465

5.3 INCIDENCE MATRIX

The *incidence matrix B(G)* of an undirected graph G with n nodes and m edges is an n x m-matrix (b_{ij}) of zeroes and ones, where n is again the number of nodes and m is the number of edges. If the nodes are labeled *1,2,...,n* and the edges are labeled $e_1, e_2, .. , e_n$ then entry b_{ij} is *1*, if edge e_j meets node i, and *0*, if not.

In the case of digraphs we have to distinguish between outgoing and incoming edges in a point and set $b_{ij} = +1$, if edge e_j starts in point i, $b_{ij} = -1$, if edge e_j ends in point i, and $b_{ij} = 0$ otherwise.

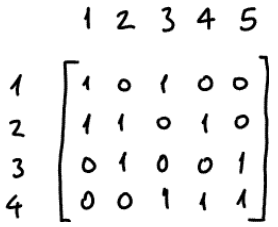


Figure 468: The incidence matrix of the graph in Figure 464

5.4 REPRESENTATION OF GRAPHS AS LIST

Graphs are relations and can be stored and manipulated as such, using the methods described in chapter 16. Undirected graphs give one incidence relation from edge to nodes

incidence:: edge -> {nodes}

from which the adjacency relation can be derived.

adjacent = (incidence . incidence $^{-1}$)\I

Note: the composition with its inverse includes also the connection of a node with itself. This is subtracted at the end.

For directed graphs, we start with two functions start and end. The adjacency is then just the composition of *start.end.*

start, end:: edge -> node

6. OPERATIONS OF A GRAPH ALGEBRA

The graph algebra consists of constructors to construct an empty graph and to insert a node or an edge into a graph and observers

to detect incidence and adjacency. Consistency constraints may be:

- Only nodes that exist in the graph can be connected and the deletion of a node deletes also all the edges incident with this node.
- Incidence is split in two relations: start and end of edge, which means all edges are internally directed.

Operations to construct the graph are:

```
emptyGraph:: g
insertNode :: n -> g -> g
insertEdge :: n -> n -> -> g -> g
removeEdge :: e -> g -> g
removeNode :: n -> g -> g
```

Observers are:

```
edges :: g -> [e]
nodes :: g -> [n]
incidence :: :: n -> g -> {e}
adjacencey :: n -> n-> g -> Bool
adjacentNodes :: n -> g -> [n]
connectedEdges, connectedEdgesStarting,
   connectedEdgesEnding :: n -> g -> [e]
nodeDeg :: n -> g -> Int
```

7. Operations on graphs

There are transformations of a graph into another graph. Two cases are relevant for GIS:

7.1 Line Graph

A line graph is the result of replacing every edge in a graph by a node and connecting nodes if the corresponding edges in the original graph were incident at one node. Not all graphs are the line graph of another graph.

Line graphs have applications when nodes have labels, for example in a transportation network, the nodes may be labeled with the expected delay.

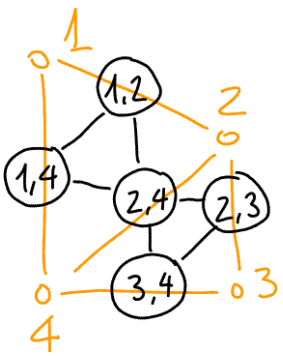


Figure 469 The line graph from the graph in Figure 464

7.2 Complement Graph

The complement graph has an edge between any two nodes where the original graph did not have an edge and no edge between nodes which were connected in the original graph (Figure 470). The complement graph has the same nodes than the original graph, only the set of edges is complemented.

7.3 Minimal spanning tree

For each graph the minimum set of edges which retain connectivity is called the minimal spanning tree. It consists of

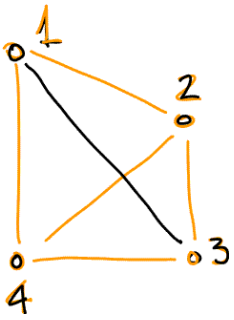


Figure 470 The complement graph to the graph in Figure 464

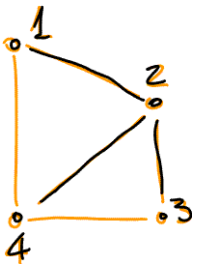


Figure 471 Minimal spanning tree

Shortest Path:
minimal sum of edge values (labels)

Edge values must be positive!

some of the edges of the original connected, undirected graph, such that all nodes remain connected (Figure 471).

8. Shortest Path Algorithm in a Weighted Graph

Determining an optimal path between two nodes in a weighted graph is one of the most important operations on a graph. There are many paths between two nodes, in graphs with cycles even infinitely many. To select from these the ones which are optimal for some criterion, is a question that occurs in many applications. For example, we may ask for the shortest path between two nodes, or the path which gives minimal travel time.

An application of the 'shortest path' algorithm uses an optimality criterion, which requires positive labels in the graph and it minimizes the sum of these label values. The prototypical shortest path takes the length of the edges as labels and determines the path with the minimal sum of edge length.

Shortest path is an example of a large set of problems, where a solution with a minimal sum of some property is searched for. It is the discrete case of Fermat's principle, which stated that the path of light in a medium is the path that takes minimal time; from this follows Snell's law. (Fermat's principle has been generalized since).

In general, problems that ask for the selection of a set of elements resulting in some minimum value can be computed by producing all possible combinations evaluate them and select the one with the minimal value. For many practical applications, such an approach is not possible, as the number of possible combinations is very large—consider all the possible ways to get from A to B in a city—and to produce all of them to identify the one that is fastest is not practical. Dijkstra’s algorithm to determine the shortest path in a graph is an example how we can produce the candidates in order such that the shortest one is found initially without exploring all the other possibilities.

8.1 Dijkstra’s algorithm

Dijkstra has published an algorithm for the determination of the shortest path in a graph where the labels are the length of the edges(Dijkstra 1959). The algorithm requires that all labels are positive (non-zero), which is automatically fulfilled if the labels give the length of the edges; this is why it is known as 'shortest

path' algorithms. The location of the nodes is not relevant for the algorithm.

The algorithm is explained in terms of ‘cost’ to reach a node. Cost is summing the weight at the labels; it is an abstract concept of accumulation of the labels and can be seen as utilization of some resource along the path. The algorithm identifies the path with the minimal cost.

The algorithm starts from the given start node and a cost for this node of zero. There starts an **expansion** step: For all nodes in the graph, obtain the cost of moving to them—a cost value *maximum* indicates that there is no direct connection. The cost of reaching the current node plus the cost of moving along the edge gives the cost of reaching the next nodes. The list of nodes with the cost of getting there and the edge traveled is added to the current list of reachable nodes. If a node is reached that was reached before, then only the path with lower cost is retained. Then the expansion step is repeated for the node which is currently least expensive to reach: If it is not the desired target node, then this node is expanded according to the procedure just explained.

The shortest path search as proposed by Dijkstra searches from the given node in circles of equal cost around the start node till it hits the target. The expansion goes in all directions, even the direction opposed to the target (Figure 472) because Dijkstra's algorithm works independent of the embedding of the graph in 2d space and uses no concept of 'direction' or 'direction to the target'; it considers only the values of the labels.. The next chapter gives an algorithm which follows the direction to the target, but it requires an embedding (i.e. coordinate values for the nodes).

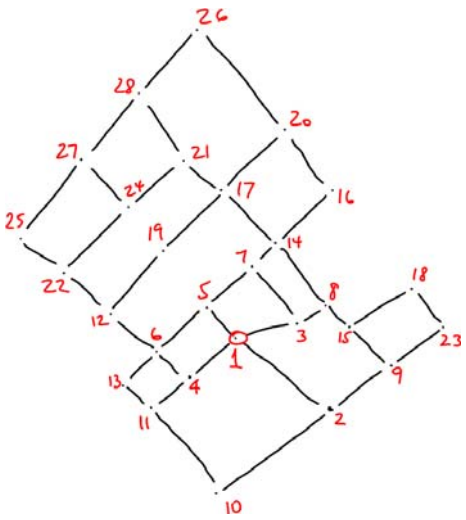


Figure 472: order of expansion

8.2 DESCRIPTION (FOLLOWING KIRSCHENHOFER(KIRSCHENHOFER 1995))

The following algorithm determines for a fixed node [ ] the *distances d(x,y)* to all other nodes *y*, i.e., the lengths of the shortest paths between *x* and *y*. Furthermore it constructs a function *p* in such a manner that starting from any node *y* the sequence *p(y), p(p(y)), p(p(p(y))),...* of nodes determines a shortest path connecting *y* with *x*.

In the algorithm the cost for connections between two nodes which are not connected are set to infinity; when later selecting the node with minimal cost so far (step (2)) and when a new cost to a node is computed, a cost value of infinity excludes these non-connections. In an implementation, these connections are not considered, but this leads to more complications in the description.

The algorithm is in 4 steps:

"(1) Initialize: Set

$$W := emptyset, U := V,$$

$$l_U(x) := 0, l_U(y) := \infty \text{ for all } y \neq x, p(y) := * \text{ for all } y \in V$$

(2) Determine the minimum of $l_U(y)$ over all $y \in U$

Choose a node [ ] such that $l_U(z)$ equals the above minimum.

Set $d(x,z) := l_U(z)$

(3) Set $W_1 = W \cup \{z\}$, $U_1 := U \setminus \{z\}$,

as well as for all $y \in U_1$

$$l_{U_1}(y) := \min(l_U(y), l_U(z) + w(z,y)).$$

If in this last expression $l_U(y) > l_U(z) + w(z,y)$

then set [ ].

(4) If [ ], the algorithm terminates.

If [ ] for all [ ], then the graph is not connected, and there exists no path from *x* to the nodes in *U*.

Otherwise set [ ] and return to step (2)."(Kirschenhofer 1992, 159)

Let us consider the following network with cost function defined on its edges (Figure 474):

After initializing we have

[ ].

After having passed steps 2,3 for the first time we have

[ ]

as well as the following new values of the *l*- and *p*-functions

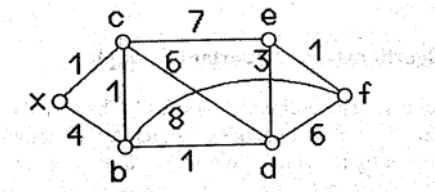


Figure 474: The initial graph

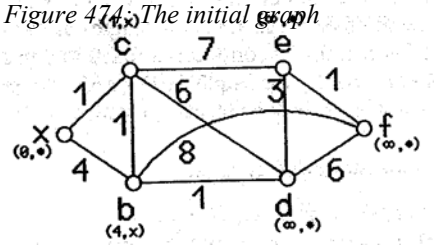


Figure 473: After the first expansion

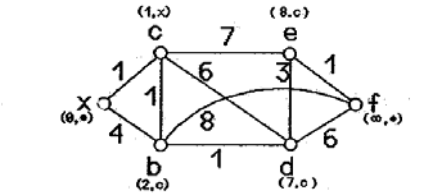


Figure 475: After the second expansion

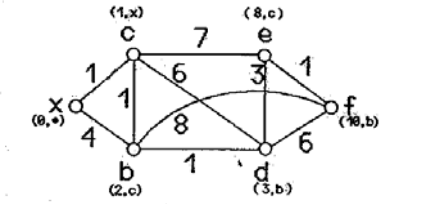


Figure 476: After the third expansion

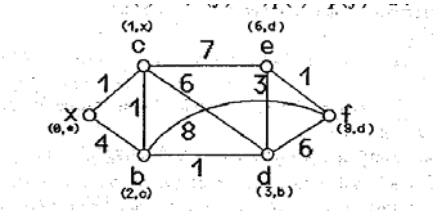


Figure 477: After the fouth expansion

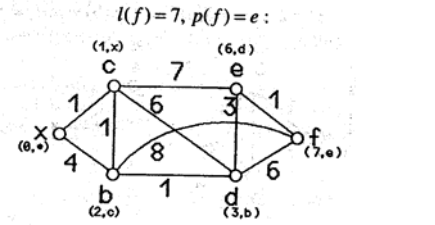


Figure 478: After the fifth expansion

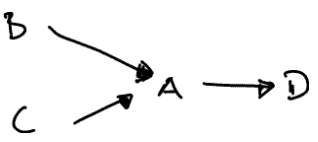


Figure 479: Example direct graph

If we attach to each node *y* the array *(l(y),p(y))* we get Figure 473.

After the second run through steps 2 and 3 we have

and the new values

(Figure 475).

After the third time

as well as

(Figure 476).

The fourth time yields

and

(Figure 477).

After the fifth time we have

as well as

(Figure 478).

Finally the sixth time yields

and the algorithm terminates.

All shortest paths between *f* and *x* have length 7, and the walk

is a shortest path of this kind.

8.3 Shortest path in a street network with One Way Streets

Street networks have one way streets, but not all streets are one way (Figure 480). We could either represent the network with all directional edges and have two one directional edges for every two-way street or two have an oriented graph with two kinds of edges: two-way edges and one-way edges.

The difference when computing a shortest path in a street network with one way streets is only in the operation to determine the nodes which are connected to a given node

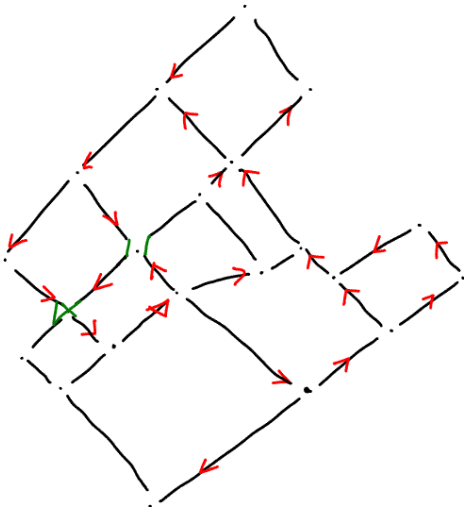


Figure 480: City streets—mostly one-way with some turn restrictions (the streets around TU Vienna!)

Figure 481: A detail from the above street network with turn restrictions shown

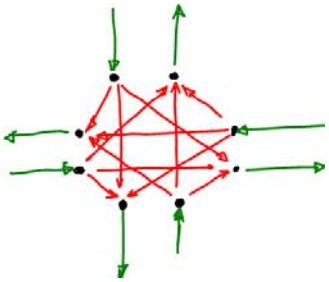


Figure 483: A full turning graph

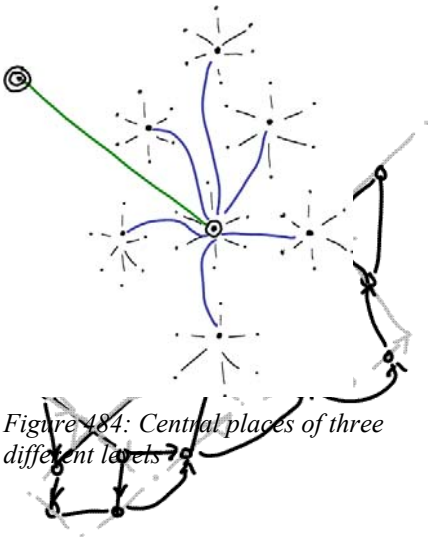


Figure 484: Central places of three different levels

Figure 482: The directed line graph corresponding to a part of Figure 482

(function *connectedNodes*). In Figure 479only the nodes *B* and *C* are connected to *A*, *D* is not connected to *A*.

8.4 TURNING RESTRICTIONS

Police regulations in cities often limits the turns possible at an intersection. Street networks representing driving operation in the real world must also represent these turn restrictions, modeling situations where signs 'no left turn' or 'no right turn' are posted. Turn restrictions require that the internals of a node are again a small network, describing which connections are possible. Figure 483 gives an intersecting where all turns are possible; the small edges in the turning graph can be given waits, to indicate how much time is lost in waiting and turning. In many real situations, only few turns are possible (Figure 481)! Alternative to turn graphs one can transform the street graph to a line graph, which represents the connection between edges in the original graph and compute the shortest path in the line graph.

9. HIERARCHICAL ANALYSIS OF A NETWORK

Space displays a hierarchical structure; best known is the political subdivision of a country in states, and the states in counties, which in turn are subdivided in towns (Figure 457). Christaller has pointed out that such a structure develops some regularity due to human behavior(Christaller 1966).Each level of central services - from providing daily supplies to services which are used once a month, once a year or once a lifetime – requires larger service areas to collect enough business to survive; therefore centers of different level occur in different distances. Christaller investigated the relatively flat area between Vienna, Munich, and Frankfurt and found a surprising regularity (Figure 484).

How to detect the dominant connections in a network of nodes? How to form the hierarchy? Given a network structure of relations between towns, where each town is connected to each other to some degree. Assume that we have a matrix that gives the strength of the connection between any two towns in an area, for example the number of phone calls exchanged between the two nodes. Following a suggestion by Nystuen and Dacey(Tinkler 1988, 265), we identify for each node the total strength (i.e., the total of calls connecting this town) and the strongest link. The total strength orders the town by strength. A node is independent, if it's largest flow goes to a smaller node, a

node is subordinate (or a satellite city) if its largest flow goes to a larger city(Tinkler 1988, 266). This gives a graph that reveals the structure of relations between the centers in the area.

10. SUMMARY

Graphs Are about Transitivity

The concept of transitivity that is motivated by the connection in a path network and is experienced in many contexts is the core of graph theory. Equivalence classes, transitive closure and fixed-point operations have become fundamental and most powerful ideas for the definition of semantics of functions; they are motivated by graphs.

a < b && b < c => a < c	*Transitivity*
f (n) = f (f (n))	*fixed point*

Transitivity in connection leads to the concept of a path, as a sequence of connected edges and—given different paths between two nodes—to the request for the shortest path between two nodes; the computation of a shortest path in a graph is a discrete form of the calculus of variation. Dijkstra's algorithm shows how this problem can be solved efficiently.

REVIEW QUESTIONS

- How can you compute the outdegrees resp., the indegrees of the nodes from the entries of the adjacency matrix?
- Explain the purpose of the shortest path determination and list several applications (other than in a street network).
- Explain the concept of Dijkstra's algorithm. How is the search progressing?
- Why is Dijkstra's method not effective to find a shortest path in a regular grid?
- Does Dijkstra's algorithm work for graphs with negative weights? Why not?
- Why does the *minPath* operation not find the best solution if trains on an edge can overtake each other? How can it be improved?

Chapter 27 LOCALIZED NETWORKS

Graphs for which the location of the nodes in space is known have many applications in geography. Street and river networks are perhaps the most visible, but also the airline and the railway networks are localized. Transportation in general follows networks and the cost is—in first approximation—proportional to the distance traveled. To determine the shortest path in an embedded network can be answered more effectively than with the shortest path algorithm of Dijkstra, which was shown in the previous chapter.

The embedding of a graph in space by assigning a coordinate pair to each node leads not only to improved efficiency when computing the shortest path, but embedded networks show specific forms, which we perceive as *gestalt*. This is the result of the processes shaping the network. Spatial analysis methods can differentiate the processes which were at work!

Graphs are invariant under a class of transformations larger than topological transformations: two graphs are equivalent if they have the same connectivity between nodes. Planar graphs are graphs which can be embedded in 2d space such that no two edges cross. Topological transformations leave planarity of a graph invariant.

1. OPERATIONS FOR EMBEDDED GRAPHS

Networks in the real world are embedded in *2* or *3d*-space. The cost function we optimize in shortest path algorithm is real world distance along the edges. To construct an embedded graph, position information must be associated with each node, i.e., we have a function

nodePos :: n -> Coord

which returns for each node its position in space. Metric operations between points translate to functions with the node identifiers as inputs: distance between nodes, bearing between nodes, etc.

```
distanceInGraph :: graph -> nodeId -> nodeId -> float
bearingInGraph :: graph -> nodeId -> nodeId -> angle
```

These functions are the combination of a function to find the coordinates for a given node identifier and the function to compute distance or bearing from coordinates (*distance* resp.,

bearing above). Edges can only be inserted when the two nodes were stored before and the distance between the nodes is automatically computed and need not be stored.

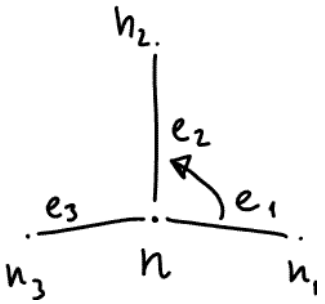


Figure 485: e_2 is the next edge around n after e_1

2. Order of Edges around a Node

For a planar graph, the order of edges around a node is fixed and remains the same under topological transformations. In an embedded planar graph, it is often necessary to find the next edge to a given edge (Figure 485). We have already used this function to determine for a point in which triangle it falls. Triangulations are just a special case of a planar graph (chapter 25xx).

The computation to determine for a given edge which is the next edge incident with this edge in positive turning direction, requires several steps, requiring retrieving all incident edges with the node, determine their bearing and sort the edges:

```
Find next edge at node (e, n):
1. retrieve all edges starting or ending at a given
node
2. compute bearings for each of them
3. sort edges by bearing
4. find the given edge
5. return the next edge from sorted list
```

This operation is time consuming and if used often, it is advantageous to store the order of edges around the node as a function, such that the next edge to a given one can be retrieved quickly (Figure 487). Care is necessary, as this introduces redundancy in a subtle way and opens a door to possible inconsistencies; the next sections show how these

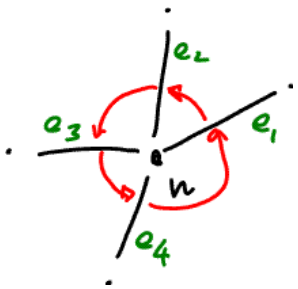


Figure 486:Orbit around node

3. An Algebra to Store Cyclic Sequences: The Orbit Algebra with the Operation Splice

Cyclic sequences in which a function *f* produces after some repetitions the initial value again ($f^n(a) = a$) are called orbits. The function *next* which gives for an edge the next edge around this node is going through a cycle ($next^4 = id$ for the node in Figure 486). Guibas and Stolfi have shown an algebra for orbits that is useful to maintain the orbits of edges around a node(Guibas and Stolfi 1987).

Orbits are graphically represented as chains of links, as in Figure 487.

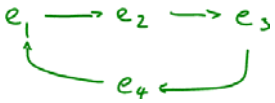


Figure 487: An orbit

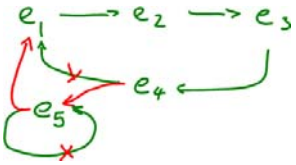


Figure 488: The operation splice (e4, e5)

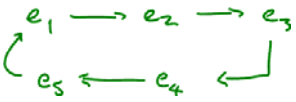


Figure 489: The result of splice (e4, e5)

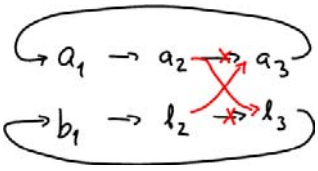


Figure 490: Merging two orbits: splice (a_2, b_2)

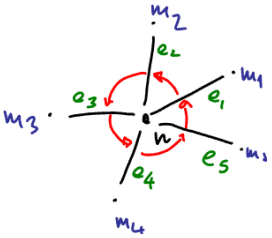


Figure 491: Orbit around n with additional edge e_5

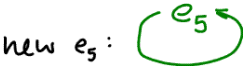


Figure 492: A new orbit with a single element

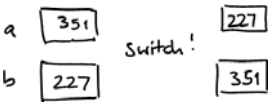


Figure 494: The operation switch

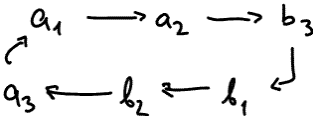


Figure 495: The result of splice (a_2, b_2)

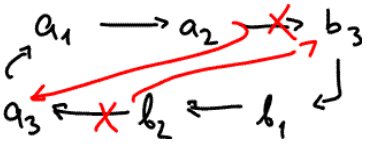


Figure 493Splice (a2,b2) applied again produces Figure 490

One might expect for orbits operations to insert an element, to remove an element, to merge orbits, etc.—similar to the semantics of sets. This is a possible, but not elegant, approach, because only one essential operation needed, called *splice* together with the trivial operation *new* to make an orbit with a single element. Inserting a new edge e_5 to *n* explains the operations (Figure 491):

The operations splice has two arguments *m* and *n* and switches the value of *next* for them. Splice is a parallel assignment, a switch (Figure 494) of two pointers {Knuth, 1973 #2467}.

Before: next m = o; next n = p
splice m n
After: next m= p; next = o

This "merges in" an orbit after the indicated position, with the starting element the *next* of the second argument. Despite this 'asymmetric explanation' is the operation commutative, *splice a b = splice b a*. Splice is its own inverse: *splice a b. splice a b = id*, as shown in Figure 493.

Orbits represented in this form are called linked lists in computer science. Figure 490 gives an example for splicing two larger lists; the result is shown in Figure 495.

4. REPRESENTATION OF EDGE AS HALF-EDGE

Edges are incident with two nodes; we have used so far a representation of pairs *e = (n,m)* and stated that the order of the nodes is not relevant and *(n,m)=(m,n)*. In a computer implementation, the two representations are differentiated and we will need to keep track which end of the edge we are interested in. For example in Dijkstra's shortest path algorithm one needs to obtain all the edges emanating from a given node and then the nodes these edges lead to with the cost of traveling along the edge, but the edges are stored arbitrarily (Figure 496) and need be organized to have the node we are interested in the first place. This is inconvenient and a better solution is to split an edge in two half edges.

A half edge originates at one node and is linked by the function *sym* to the other half edge; half-edges around a node are linked by *next* (Figure 497, Figure 500). These 3 functions *origin*, *sym* and *next* can be stored as relations in a database and give, for example the adjacent node from a half edge by composition *sym.origin*. Starting with a node n and using the converse

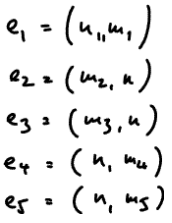


Figure 496: List of edges around n in Figure 491

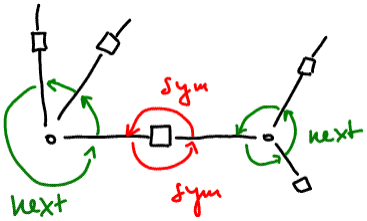


Figure 497: Two half edges with pointers to next edge around node and to other half edge

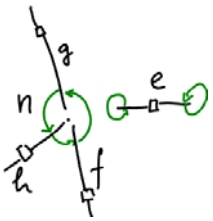


Figure 498: Node n with edges f g h and new edge e

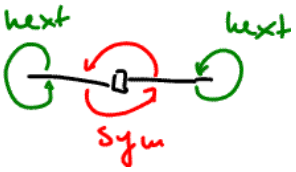


Figure 499: A new edge

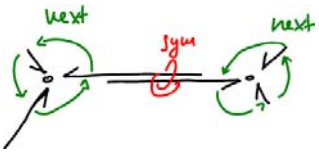


Figure 500: Alternative visualization of half-edges

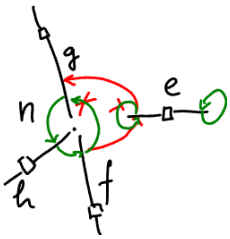


Figure 501 Splice (e,f)

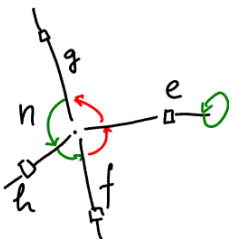


Figure 502: Result is node n with orbit f, e, g, h

relation *origin'*, we obtain all adjacent nodes by *map (origin. sym) . origin'* ; the result is a list of nodes because *origin'* returns a list of half-edges. An operation *newEdge* produces the two half-edges with orbits around their ends and the sym functions (Figure 499).

5. Operations to Maintain Graph

To maintain a graph with orbits around the nodes requires that all operations that change the graph not only add nodes or edges but also update the orbits around the nodes. One operations to insert a half-edge into an orbit would be sufficient, because it could also be used to remove a half-edge from the orbit (remember: switch is its own inverse). An operation to insert a node into an edge is added for symmetry.

5.1 Insert edge in orbit around node

Given a node n with an orbit *f,g,h* and a new edge *e* (Figure 498). The new edge should be inserted between *f* and *g* into the node *n*. This is achieved with the operation *splice (e,f)* (Figure 502) and the result is as desired an orbit *f, e, g, h* around *n* (Figure 502). Observe that the arguments to the splice operation are the two half edges, such that each is inserted after the other in their respective orbits. Applying *splice (e,f)* again removes the edge *e* from the orbit. The same operation is used to connect the other end of the edge to the adjacent node.

5.2 Insert node to split edge

Inserting a node in an edge produces also a new edge. This can be achieved with one *newEdge* and 3 *splice* operations. The first *splice* removes the edge from the node (Figure 503, Figure 504) and then two *splice* operations close the orbits around the old and the new node (Figure 505). It is possible, to achieve the same effect with only two splice operations, if the *splice* is applied to the sym functions (the result are then two edges *e1,d1* and *d2,e2*).

5.3 Follow a cycle

The functions introduced can be used to follow in a graph around a minimal cycle (Figure 506). Start with a half-edge, say e_1 at m. $i_2 = next\ (e_1)$, which means we follow the cycle clock-wise. To get the other half edge is $i_1 = sym(i_2)$, then again the next edge

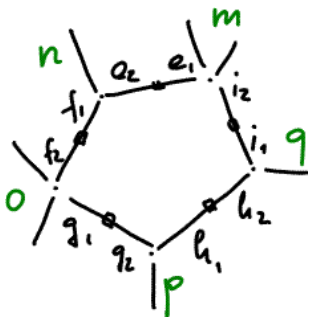


Figure 506: A graph with a cycle m,n,o,p,q

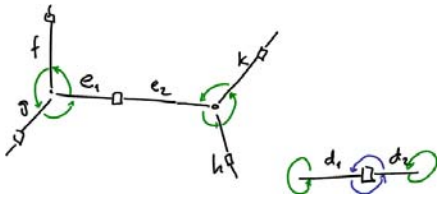


Figure 503 Insert new edge d after e

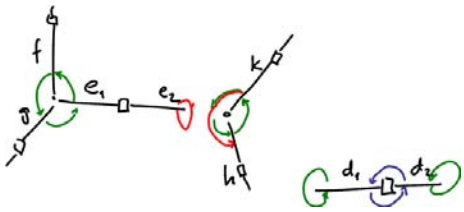


Figure 504 Splice (k,e_2) opens the orbit

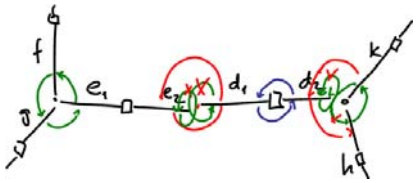


Figure 505 Splice(d1,e_2) and splice (k, d_2) inserts edge

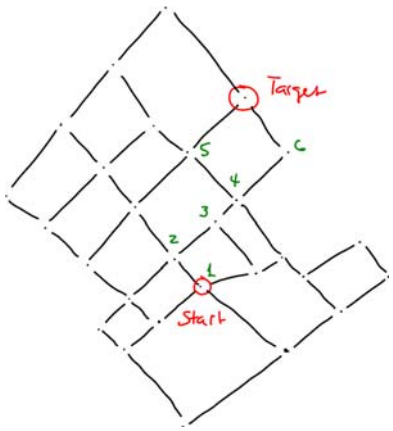


Figure 507: The order in which nodes are expanded from start to target

around the node h_2 = *next* (i_1) etc. an alternative chain of sym and next operations get us back to e_1.

6. SHORTEST PATH IN AN EMBEDDED GRAPH

In an embedded graph, no labels for the length of an edge are necessary, because the distance can be computed from the coordinates of the points. Dijkstra's algorithm shown in the last chapter works for embedded graphs, but is not the fastest method. In an embedded graph, directions are fixed by the location of the nodes and a shortest path algorithm should progress primarily into the direction from the source to the target. The A* algorithm uses this additional information from the embedding of the graph and is therefore more efficient. In Figure 507 the order in which nodes are expanded are marked and in this example only one 'unnecessary' node not on the shortest path is expanded; in Dijkstra's algorithm, the target would have been only found after 20 expansions (figure previous chapter xx). Geometrically, an A* algorithm searches similarly than the Dijkstra algorithm, but it adds to the cost of each node the cost to reach from there the target—this gives some directionality to the search.

After an expansion step we know for each node the cost to arrive there from the start; Dijkstra's algorithm expands then the node with minimal cost. In an embedded graph we can also compute the Euclidean distance to the target, which gives the least cost to reach the target from there—any path in the graph will be longer. When selecting the next node to expand, we do not select the one with the least cost to arrive to, but add to each node the cost from there to the target and expand the one with the smallest total for cost to arrive here plus minimal cost to the target.

Figure 509 shows the graph from the previous chapter (figure xx) with a few nodes and edges added. In this graph, Dijkstra's algorithm needs to expand 8 nodes (roman numerals). The A*-algorithm shown in Figure 508 expands only 5 nodes. The estimates of distance to target which are added to the cost 'so far' keep the expansion clearly in the right direction!

If we minimize some other values then additional information for the edges may be necessary; for example, one might want to minimize travel time. The time necessary to drive along an edge

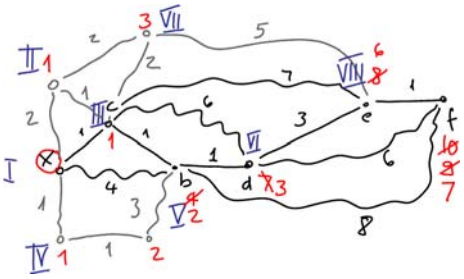


Figure 509: The graph from the previous chapter embedded in Euclidean space and with some additional edges.

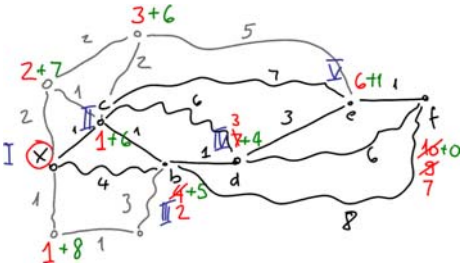


Figure 508 The A algorithm expands only the 5 nodes on the shortest path!*

depends on the length of the edge, i.e., the distance between the nodes, divided by the speed; the average speed for an edge depends on the width of the road, the radius of the curves, etc. To store the design speed, width of the roadway, etc. may be stored as a label with the edge and then compute for each edge the time it takes to travel along it.

The A* algorithm can be used whenever we look for a minimal path in a network and have a method to estimate the minimal cost to reach the target from a given node. If the actual cost later is larger, we still find the optimal path (but not, if the actual cost would be less than the estimate!). For example, fastest path is often desired, where each edge has a label with length and an expected speed. To estimate the travel time from a node to the target we calculate the Euclidean distance divided by maximum speed; this is a minimum and the actual time will be larger, therefore it is an acceptable estimate for the A*-algorithm.

The change in the code from Dijkstra's algorithm to A* is minimal: when selecting the next node for expansion, one has first to add to the cost to reach the node the (minimal estimated) cost to reach the target; then the node with least total cost is selected for expansion.

7. Linear Reference Systems

An embedded graph provides a reference system for points. Points along the edge can be identified by their distance from the start point of the edge (Figure 510). This is a natural parameterization of the edge. Functions to calculate the coordinates of a point along a line have been introduced (see parameterization of *1*-simp, chapter 23xx).

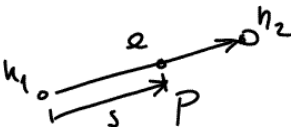


Figure 510: A linear reference

A linear reference is composed of an oriented edge (or an identifier for it), and a distance s from its origin; this determines a point on the edge (Figure 510). It is possible to add a lateral position l and a height h to identify a point in 3d related to the edge (Figure 511).

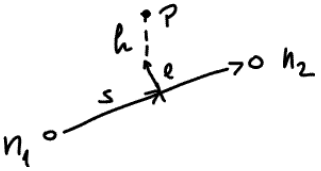


Figure 511: A reference to a 3d point

7.1 Mileposts

Mileposts along a highway (photo xx) use this kind of reference system, but they measure the distance along a path consisting of many segments. This method is convenient and widely used, by road administration, waterways and railways. Mileposts are

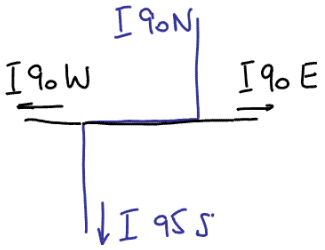


Figure 512: One street segment is part of 2 numbered interstate highways.

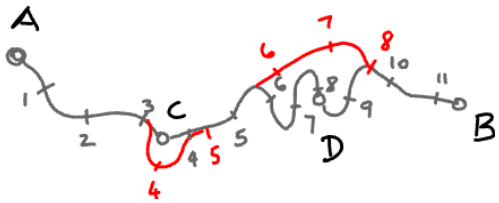


Figure 513: Original road and a new detour at C and a shortcut at D

different from linear reference systems along an edge in a graph. They use not a single edge but a longer 'route' as the unit along which the reference length is measured. This introduces a number of problems:

- Sometimes a single road segment is part of more than one route (Figure 512); the same point has then 2 different mileposts!
- Routes become shorter or longer by construction of by-pass or short-cuts.

Consider the original road mileposting in Figure 513. There are about 11.5 km from A to B and mileposts are set along the road. Later, the road is improved to avoid the town C with the narrow passage and the town D on the hill. Avoiding C creates a longer (but hopefully less congested) road that reaches the previous road at milepost 4.3 (but with its own milepost 5)—the same mileposts are now used twice! Shortening the road at D makes the mileposts between 8 and 9.6 to disappear. The clean solution to redo the mile posting from A to B is avoided because it would require to physically relocate the existing mile posts along the road, but it would also invalidate all references to locations along the road using mile posts—for example in all the legal documents that pertain to the posting of street signs, traffic restrictions and also accident reporting. In practice, the doubly counted new miles at C are marked and it is hoped that not too much confusion emerges!

7.2 Street addresses as linear reference system

A street address, with street name and number, is a form of linear reference systems – provided the building numbers are distributed along the street in increasing order. In Europe, many cities number buildings from the city center outwards alternatively on both sides of the street, such that odd numbers are on the left side and even numbers on the right side. If the building numbers on street corners are known, then other building numbers can be interpolated. This gives an approximate reference system. In America, each block starts on an even 100, with many numbers missing. This allows even more precise localization with very little precise data (Figure 515); it was used for DIME (dual independent map encoding (Fagan and Soehngen 1987) and carried over to the TIGER files of the U.S.G.S. For each street segment, the address range on each side

Many countries use regular distribution of street numbers along a street, but one must not assume that this is universal!

of the street is given; they were established to support the decennial census operations (http://www.census.gov/geo/www/tiger/chapter5.asc).

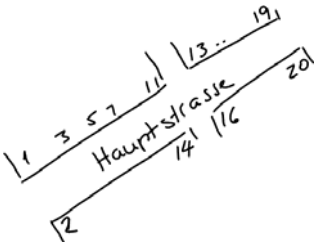


Figure 514: Typical street numbering pattern

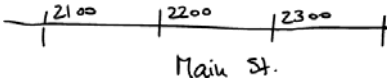


Figure 515: Regular numbering allowing 100 numbers for each block

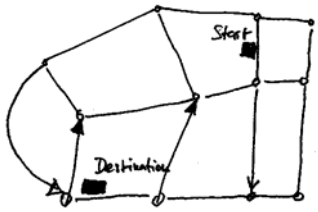


Figure 516

8. SHORTEST PATH BETWEEN POINTS ON EDGES

Real world of navigation is not restricted to operations between nodes in a graph; when we search for a shortest path between two buildings, we start on an edge between two nodes and the destination is again on an edge between two nodes (Figure 516). A translation of a building street address to a linear reference on the edge can use the interpolation method suggested in the previous section. In order to use one of the shortest path algorithms discussed, the start and the destination must be inserted into the graph as nodes with the function to split an edge shown before (section 6xx).

8.1 MILEPOSTING

8.2 MILEPOSTING AS LINEAR REFERENCE SYSTEM

The mileposting method can be seen as a coordinate system, namely the coordinate system for a line with a single coordinate (one degree of freedom). This is used practically: everybody knows the milepost along a highway (photo).

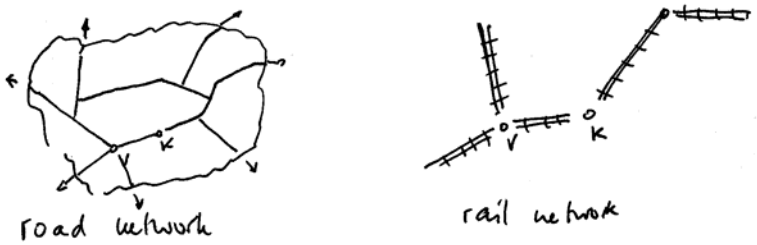


Figure 517: Southern Austria: (a) highway network, (b) railway network

9. OVERLAY OPERATIONS ON GRAPHS

Two graphs embedded in the same coordinate system (Figure 517) can be combined. For example the highway network and the rail network can be combined in a single graph. (Figure 518). The integration uses the same operations as used to integrate two simplicial complexes (remember, graphs are 1-complexes!).

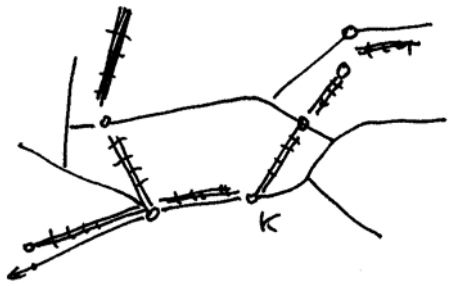


Figure 518: Combination of highway and railway network

The resulting combined graph maintains labels for each edge with its origin—is it a rail or a road edge. Assume that the edges are labeled with travel time, one can now ask what is the fastest way between two points, allowing driving or using the railway, or even mixing the two; so-called mixed mode transportation. An

application is a trip from the countryside to a major city, where we use multiple modes of transportation: first with the car to the railway station, then with the rail, then metro, and ultimately streetcar to our destination. This requires combining 4 different graphs, for the road, rail, metro, and streetcar network. In the combined graph the shortest (fastest) path can be determined.

10. PLANAR EMBEDDED GRAPH

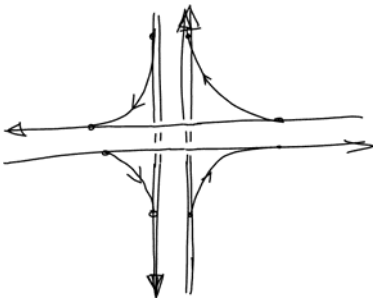


Figure 519: A typical highway intersection

Not every real world network is represented by a planar graph: streets may overpass each other. This is sometimes described as 'grade separated crossings'. If the graph is stored as a planar graph, then 'grade separated crossings' must be marked, because they do not allow turning. As an example, look at the standard highway exit: it is certainly not a planar graph (Figure 519). Alternatively, the network could be maintained as a non-planar graph with intersection of edges that are not nodes, but this is not the current preference of standards, which usually assume an underlying structure of a (simplicial) complex.

11. OPTIMIZATION OF A NETWORK

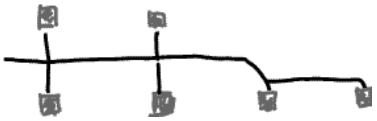


Figure 520: Minimal length connection of 6 buildings to a water source

In many cases a given set of nodes must be connected to optimize some criterion. Connecting two nodes with the shortest path has been discussed extensively, but other situation with other optimization goals are common and they often lead to specific gestalt of a network. Given a graph, it is often possible to identify the kind of network represented; a street network in a city or overland looks different and a stream network is easily recognized.

11.1 SHORTEST PATH NETWORK

Assume that a number of buildings must be connected to a water distribution system (Figure 520). In the abstract the problem is to connect a number of service points with a mimimal length network. The network with the shortest path is characterized by interior angles of 120 degrees. Consider the three prototypical situations: three collinear nodes, three nodes with an angle of less than 120 degrees and 3 nodes in general situation (Figure 521). For a larger set of nodes, the edges all meet at internal (additional nodes) under 120 degree angles (Figure 522, Figure 523), but practicality may dictate other choices: distribution networks in cities are typically buried in the street and connections to buildings are then at right angles.

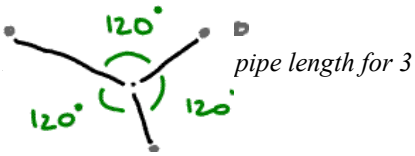


Figure 522: Typical connection of 3 nodes in general position

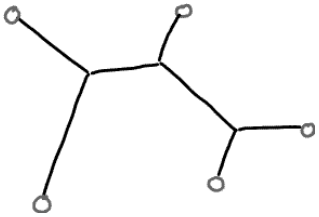


Figure 523: Network with minimal length

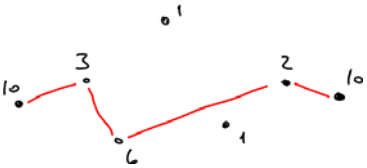


Figure 524: A network which optimizes net income (construction cost-income). The figures for the nodes give their potential income

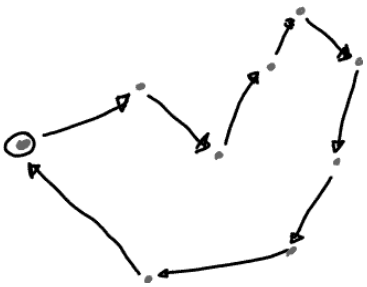


Figure 525: Traveling salesman

11.2 Maximal income

Consider constructing a railway or a highway, where the cost to link to a node must be compared to the revenue obtained from linking to this node. Connecting all the nodes is not the solution that maximizes the income, as the following example (from Abler, Adams et al. 1971, 275) shows (Figure 524).

11.3 Traveling salesman problem

This is a "famous" problem in computer science, not for its practical relevance but for its computational complexity. A salesman has to make a number of calls to clients, located at different points, and return at the end; what is the shortest path he can take? In which order should he visit the clients?

In the abstract: determine the shortest path connecting a set of nodes. This problem has been extensively studied and it has been proven that no fast algorithm to certainly identify the shortest path is feasible. Several reasonable methods to find a close to optimal solution are known (Figure 525).

A variant of the traveling salesman problem is the "Car pool" (Paul Revere) path. Find a path from a start to a destination, linking all intermediate nodes in any order to obtain the shortest path (Tinkler 1988) (Figure 526).

11.4 The form of the network reveals what was optimized

Studying the form of a network allows us to identify which function was optimized. Internal angles of 120 degree indicate that the cost of construction was minimized, generally, a shortest set of edges was identified (Figure 523). If the graph is completely connected, then convenience for use was optimized (Figure 526). A hierarchical arrangement shows connections from one point out. A close path (cycle) reveals an attempt to serve several points and return back (Figure 525).

Review Questions

- What are orbits?
- Why is splice its own inverse? What does this mean? Do you know other operations with this property?
- What is the difference between A* and Dijkstra's algorithm?

Figure 526: Optimal path for a car pool

- When can A* be used?
- What is the effect if the estimate of remaining effort in A* is wrong?
- How do we determine the shortest path between two points located on a segment (not a node)?
- When do we need Identifiers? Why?
- Exercise: find algorithmic solutions for all the optimal network problems.
- Apply splice (e,f) to Figure 502. What is the result?

PART TEN CELLS

Figure 527: Example map

Maps show the world described by curved lines and areas (Figure 527). Land is divided in parcels, each owned by one person (Figure 528). In this part the representation of curved lines and arbitrary subdivisions of space created by them are discussed. It is shown how they are related to general representation of geometry in a simplicial complex (part 8).

The first chapter discusses lines as they are, for example, used to draw geographic situations on maps; collections of points are versatile and can be interpreted as linear features or bound areal features. The different interpretations of collections of points as lines or areas are related to the operations applicable to them.

The second chapter discusses subdivisions. We want a subdivision in parcels to remain such that every piece of land is assigned to one parcel and that the parcels do not overlap. Such divisions occur in spatial and non-spatial situations and are called *partition*; we will use the term *subdivision* for partitions of space.

Figure 528: Parcels as an example of a subdivision

Not only ownership of land produces subdivisions, but also the subdivision of a continent in countries, countries in provinces, etc. are subdivisions—and other examples from administration abound (school districts, church parish, etc. see figure before xx). Natural processes can lead to subdivisions: the world is divided in land and sea (Genesis xx), land is divided in watersheds. Much data is collected with respect to subdivisions and combining this data requires effective processing of subdivisions.

Assigning area to a service point is a crucial link between space and points: a hospital (point resource) serves a region (area), a grocery store is visited by clients from a block in a city. Voronoï diagrams show service areas such that every point is served by the nearest service Voronoï diagrams are the dual of the Delaunay triangulation. In the last chapter the general

approach of subdivision of space in cells is connected with simplicial complexes.

Chapter 28 CELLS: COLLECTIONS OF SIMPLEXES TO REPRESENT CARTOGRAPHIC LINES

In this chapter we deal with methods to represent maps (Figure 529). The discussion is considering the map here as a *collection of points and lines*, and ignores the complex, multi-layered structure that is communicated to a human map reader and the use of the line network as a graph to find a route, as discussed in the previous chapters (26 and 27). The chapter does also not address placement or content of labels, characters, numbers and symbols, which describe objects on a map.

This chapter generalizes simplexes to cells. The geometry of a cell can be represented by a list of coordinated points, and this list can have different interpretations: it can represent a collection of points, a line, or even an area (*0*-, *1*- and *2*-cells). These different interpretations correspond to different algebras applicable to the same representation and will be defined as transformations between different complexes.

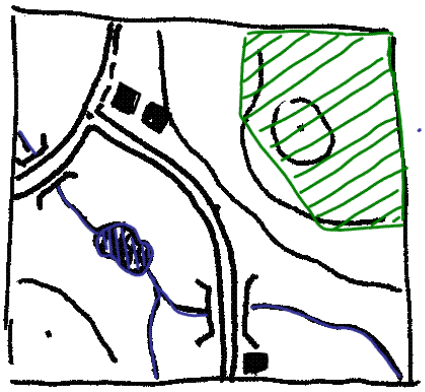


Figure 529: The points, lines, and areas of a map

1. INTRODUCTION

In the example map without symbols and text (Figure 529), we find

- points that indicate the location of points in reality (for example, points with measured height);
- lines that stand for a path between two locations, or a brook running in a valley;
- areas, which stand for a pond, a forest, or a building.

These points, freeform lines and areas are *cells*. This chapter shows how the geometric values of simplexes (points, line segments, triangles) can be combined to produce the less restricted cells.

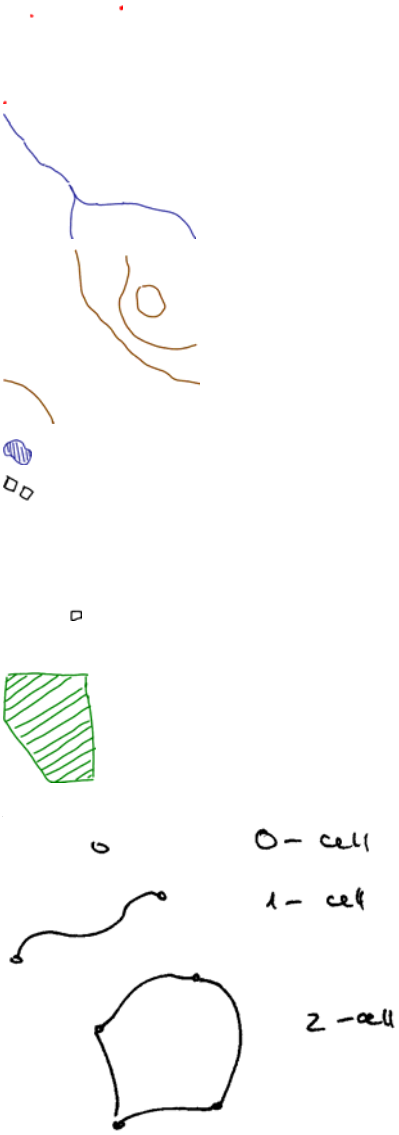


Figure 531: Cells

The curved lines on maps representing irregular forms in the terrain are approximation to the real situation. Cells can be approximated by simplices and the operations on cells implemented in terms of these approximations (Figure 530). Map graphics are representations of ideal geometric figures. The graphical representation must have an extension: a line has a width; a point is really a small area, etc. In the discussion in this chapter, we stress the ideal properties of the geometric figures and the non-ideal, realistic consideration of graphical elements is left for a discussion of visual communication and cartography. We exclude equally the labels, symbols, etc. that are for real maps.

This chapter does not address the cartographic transformation from world coordinates to map coordinates, the selection of symbols. It excludes also the simplification of lines, for example the so-called Douglas-Peucker algorithm. This is left for a comprehensive discussion of approximations.

ınd The focus of the chapter is on operations with lines. These operations depend on the interpretation, especially of the dimension of the objects manipulated. Collections of points can be just that, collections of points, but they can also be lines, closed lines or even areas. Different operations apply to these different interpretations, other operations transform between interpretations. For example, the convex hull transforms a collection of points in a single cell. Triangulation transforms a collection of points in a collection of 2-simplices.

2. CELLS

The simplices are the simplest geometric figure in each dimension; they are restricted to straight connections between the points. The restriction to straight connections can be removed to give the generalized concept of cell. Cells, like simplices, exist in each dimension; there are *0*-cells, *1*-cells, *2*-cells, etc., each topologically equivalent to the corresponding unit sphere (Figure 531).

Figure 532: Approximation of 1-cell by 1-simplices (at two different quality levels)

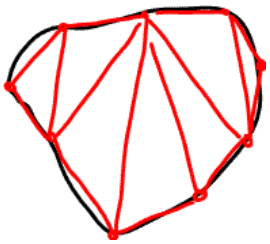


Figure 533 Approximation of 2-cell by 2-simplices

0-cells are simple points and coincide with *0*-simplex.
1-cells are curves between two *0*-cells.
2 cells consist of an undetermined number of *1*-cells as boundary.

Operations on cells are (mostly) the same as for simplices. The cells can be approximated by collections of the simplices of the corresponding dimension and operations on cells are defined by operations on the simplicial complex used to approximate them; the result of applying the operation to the individual simplices are then summed or otherwise appropriately integrated. The operations carried forward from simplices are:

- *Rank*, *dimension,* and *codimension* are defined for cells as for simplices
- *Orientation*: All the simplices forming a cell must have the same orientation. This orientation is the orientation of the cell. A cell can be *reversed* by reversing all simplices in it. (This is implemented as operations on chains).
- The *boundary* of a cell is the boundary of the chain of simplices of the desired dimension.
- *Intersection* of two cells represented as *1*-chains in a complex: determine the degree of the nodes. If the degree for any node is higher than *2* the chains are intersecting and these points are the intersection points.
- *Length* of a *1*-cell, area of a *2*-cell: the size of an *n*-cell is the sum of the size of the *n*-simplices it consists of.
- *Point in cell test* (the ordinary *point in polygon test*): the point is in the cell if it is in one of the *1*-simplices or in the *1*- or *2*-skeleton of the cell. Alternatively, determine the boundary: the point is inside the *2*-cell if it is left of the boundary (*ccw* predicate).
- Non-branching *1*-cells are linear and can be *parameterized*. Determine the length of the line as a running sum from start to end.
- *Join and meet* are computed by converting the two cells to two complexes and then computing the join or meet of these.
- *Equality*: if the 0-skeleton of two cells are not equal, the cells are not equal. Two 1-cells are equal if they consist of the same 1-simplices; two 2-cells are equal if their 2-complexes have the same boundary (but not necessarily the same 2-simplices).
- *Incidence* and *adjacency* relations: convert both cells into complexes and determine incidence or adjacency there.

Figure 534: A line approximated by points

- *Length, area, volume, generalized to size, moments.* Metric properties are in general determined by converting to a complex and then sum the metric properties of the corresponding simplices.
- *Point in cell test* is computed as a point in simplex test after conversion of the cell to a complex.
- *Parameterization* follows the same logic than parameterization of 1-simplices: Assume a function *p (s, v)* which for a simplex and a parameter value *[0..1]* returns the coordinate pair of the point. Then compute the successions of length l_i of simplices up to point *i* (for the last point *n* this is the length of the *1*-cell). Then the coordinates is obtained with a call of the function *c (v, n)*, where *v* is the ordered list of *1*-cells and *n* the number of points in the line:

$$c(v,i) = \text{if } v > l_i \text{ then } p\left(s_i, \frac{v-l_i}{\text{length}(s_i)}\right) \text{ else } c(v, i-1)$$

- *count* the number of components in a cell (operation *cardinality*).

Novel operations, which were not meaningful for a simplex, are

- Convex hull
- Triangulation

These are transformations between geometric objects of different dimension and implementable as operations on complexes.

3. Representation and Interpretation of Cells

A cell is represented as a chain in a complex. A *0*-cell is a *0*-simplex, *1*-cells are lines, approximated by a *1*-chain from a *1*-complex, and *2*-cells are regions, approximated by a *2*-chain in a simplicial *2*-complex.

A collection of points can represent just this, a collection of points, or a collection of lines, a line, a closed line or even an area (Figure 535). In each case, the complex is replaced by its *0*-skeleton, i.e. the collection of *0*-simplices it contains; for the cases considered here, the mapping from *0*-skeleton to the *1*- or *2*-complex is an isomorphism. This can be generalized beyond 2 dimensions: Any *n* points define an *n*-simplex, thus a collection of *n* points define sets of simplices up to dimension *n*. The interpretation adds rules, which one of the many sets of simplices is intended. The use of order among the points is important to select a line (collection of 1-simplices) interpretation. Convex-hull and Delaunay triangulations are

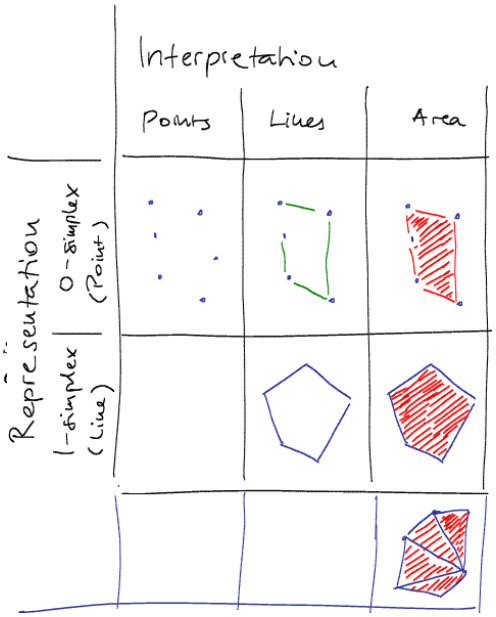
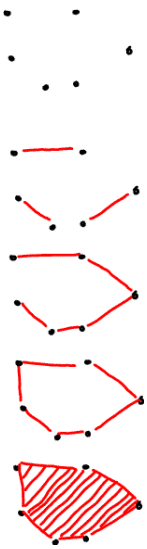


Figure 536: Representations and interpretation



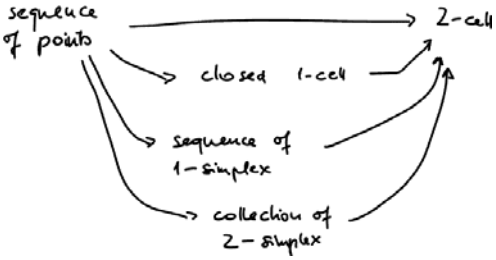


Figure 537: Different path to construct a 2-cell from a sequence of points

methods to select a definite area (collection of 2-simplices) interpretation.

An ordered sequence of points is used to represent a line or even a region (Figure 535). The operations that follow from these interpretations are geometric and must be separated from the operations to manipulate the data structure used for the representation. The uniform representation of cells and cell complexes as simplicial complex removes this ambiguity, which is otherwise visible at the user interface.

If a collection of *n*-simplexes is interpreted as an *m*-cell, then the operations of *m*-cells are applicable. This is the meaning of the phrase 'is interpreted as'. The Figure 536 shows the differences between representation and interpretation (i.e., operations applicable). If data representations are directly visible to the user or programmer, then it is left to him to select the operation applicable understanding the assumed interpretation and the required constraints. This leads to errors because it contradicts the object-oriented principle of encapsulation of behavior in a class.

We will use differentiated representations for the different interpretations and define operations for the transfer between these representations (Figure 537), even if the underlying representation is the same, namely *0*-simplices. They are internally treated as *0*-complex, *1*-complex, *and 2*-complex and eventually mapped to the same method to deal with chains, but typed differently to assure consistent use of operations. Between these different interpretations are transformations which have the general form of transforming a set of *k*-complex to an *n*-complex, with $k < n$.

4. CONVERSIONS BETWEEN DIMENSIONS

Figure 536 gives a hint to what conversions are possible. They are systematically discussed here and defined as operations on simplicial complexes.

4.1 CONVERSION FROM ORDERED LIST OF POINTS:

4.1.1 To a *0*-complex:
Convert the list of 0-simp to a 0-complex, represented as a 0-chain. Each point is a *0*-simp.

4.1.2 To a *1*-complex:
Three interpretations of the ordered list of points:
(1) Each pair of points is a separate piece of line,
(2) A line from first point through the intermediate points to the last point or
(3) A closed line connecting the last point to the first point.
Adding the first point to the end of the list converts case (3) into case (2).

In each case, a pair of points is converted to a 1-simp and then a list is formed. Complexes do not allow crossings of edges which are not nodes; if the given line is self-intersecting, then additional point may be necessary and must be marked.

Conversion may introduce additional points if the line is self-intersecting.

4.1.3 To a *2*-complex:
Insert the points into an empty *2*-complex. This forms triangle, because every point must be a boundary of a *1*-simp, which must be a face of a *2*-simp. Note: this conversion is not unique and many equivalent results are acceptable (see next chapter for a unique solution).

4.2 Conversion from 1-complex To list of Points

The conversion from the *1*-complex to a list of points is the skeleton operation (exactly the *0*-skeleton).

4.3 Conversion from 1-complex to 2-complex

Insert the *0*-simp from the *1*-complex to the (empty) *2*-complex. This forms triangles (*2*-complexes). Then insert the *1*-simps into the resulting complex; most of them will already exist, but in some cases it will be necessary to swap an edge in a quadrilateral to insert the desired 1-simp (Figure 538)

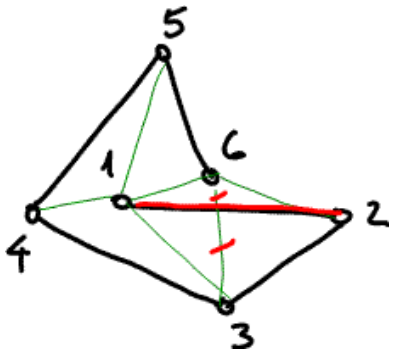


Figure 538: Example of conversion of list of points to 1-comples where swapping an edge is necessary: the edge 3-6 is replaced by 1-2

4.4 Conversions from 2-complex

A *2*-complex without the *2*-cells is a *1*-complex. The operation skeleton gives this conversion (*1*-skeleton and *0*-skeleton, or just the *0*-skeleton). Applications often request not the skeleton, but the boundary, which is a closed line and obtained by applying *boundary* to the 2-complex.

5. Implementation of General Geometric Operations

6. Special Operations for *0*-Complexes

Three operations are interesting to warrant a special discussion:

- Find the nearest neighbor to a given point,
- the convex hull, and
- the construction of a triangulation between the points.

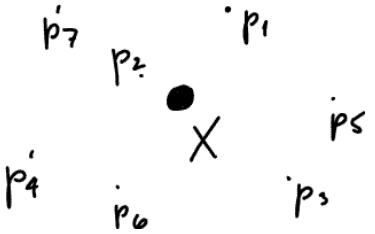


Figure 539: p_2 is the nearest point to x

6.1 NEAREST NEIGHBOR

The operation determines for a given point x the point p_n from a collection of points that is closest to x (Figure 539). A straightforward method to find the nearest neighbor is

- determine the distance from each point in the collection to x; by applying *distance to x* to each point;
- sort the points according to their distance to x;
- select the first.

Special data structures were invented to avoid searching through all points. These improved algorithms are necessary for large data collection as a GIS; but are not discussed here, because they only affect performance, but not the result produced(Samet 1990; Samet 1990).

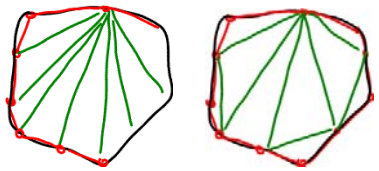


Figure 540: Two different triangulations of the same points

6.2 TRIANGULATION

The conversion of a list of points (or a *0*-complex) into a *2*-complex produces a triangulation. There are many different triangulations between a set of points possible (Figure 540). The insertion of points and incrementally producing the triangulation gives different results depending on the order of insertion. Two questions arise: is there always a triangulation for the given cell and is there a single (canonical) triangulation for a given set of points? Any 2-cell can be triangulated, but the triangulation of a 3-cell may require so-called Steiner points, i.e. additional points inserted inside the volume. A unique triangulation exists and will be shown in the next chapter.

6.3 CONVEX HULL

The convex hull of a set of points is the set of points, which are the ‘extreme’ points of the collection, such that all points in the collection are included between the connections of these extreme points (Figure 541).

The convex hull for a set of points is the smallest convex set that contains all points

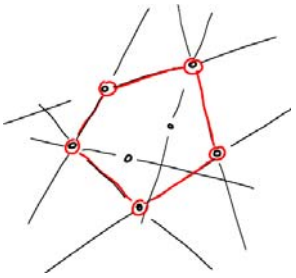


Figure 541 Convex Hull

Many algorithms are known(Knuth 1992; Edelsbrunner 2001). A simple, incremental algorithm starts with three points, which are always convex set. Add point by point: if point is inside, then return previous hull, if point is outside, add the new point and determine which of the previous points is now inside. Algorithms should avoid testing the same configurations repeatedly ("do not ask dumb questions").

Observe that the conversion of a point set to a *2*-complex and then determining its boundary is also producing the convex hull.

The computation of the complex hull has a dual problem: given a set of half-planes limited by some flats, determine the corners of the convex area delimited (Figure 542). This operation is useful for finding an area where a number of conditions, expressed as inequalities, obtain. The problem is solved by using translating the given flats into points (see duality chapter 19) and then computes the convex hull, which gives a set of simplexes. The dual of these simplices are the boundary points of the solution. This is the often used simplex algorithm to find an optimum for a set of linear constraints [ref].

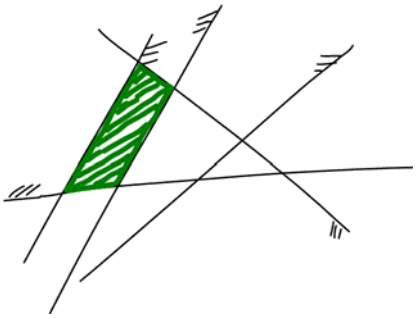


Figure 542: Area where 5 conditions are fulfilled

7. SPECIAL OPERATIONS FOR *1*-COMPLEX

A *1*-complex is a set of *1*-simplices. It can represent a single line, a set of lines, a complex line or a closed line (Figure 535). A number of tests are used to differentiate these forms.

7.1 TEST FOR CONTINUOUS AND CLOSED

A 1-complex is *continuous* if the degree of each node is 2, except for the start and endnode. A *1*-complex is *closed* if the degree of each node is 2.

The operation completion, adds 1-simplices to make the 1-cell continuous. The operation closure ads a 1-simp from the start of the 1-cell to the end of the 1-cell and converts the 1-cell to a close 1-cell, which can be interpreted as a 2-cell.

A test for self *intersecting* for a 1-cell translates to a test in the 1-complexs on the degrees of the node: A 1-cell is self intersecting if any node has a degree higher than 2 (Figure 543).

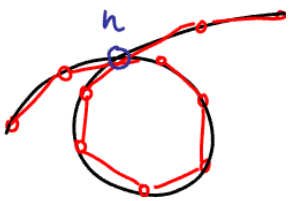


Figure 543: Self intersecting 1-cell approximated by 1-simplice. The marked node n has degree 4!

7.2 INSERTION OPERATIONS ON 1-COMPLEXES

GIS software which provides operations to manage lines as ordered sequences of points need special operations (Figure

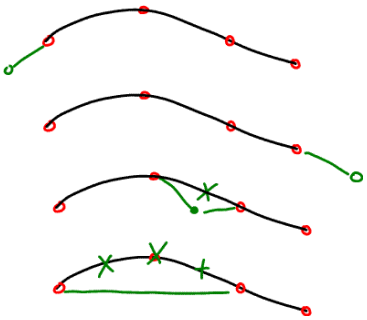


Figure 544: Operations with lines: extend at begin, at end, add point, and delete point

544). We show here, how they translate to operations on the simplicial complex:

- Extend line by one point:
  Extending a line means adding a point at the beginning or at the end – usually provided as two operations. In both cases, the new 1-simp is added to the chain, because a *1*-chain does not depend on the sequence in which the *1*-simplices are listed.
- Delete a line segment from a *1*-cell: Add the negative *1*-simp.
- Insert an element at a position (first, last, after another one). If a line should be changed to insert a point in the middle to split a line segment then the line segment that should be changed is subtracted (added negatively) to the complex and then the 2 new segments are added. To delete a point from the line, just reverse the above operation: delete the 2 adjoining 1-simplices and replace them by one.
- For directed *1*-complex: Reverse the order, find first or last element :
  reverse the simplexes one by one (i.e., multiply with -1)

8. Conclusion

In this chapter, operations with collections of points, line segments, where introduced. In all cases, we were able to relate the operations on cells to operations on simplicial complexes (chains) that are used to approximate the cells.

The similarity of the representation – graphically and in a computer – for different interpretation leads potentially to confusion: a user may not be clear what he intends to manipulate: the graphical line or the represented area? Confusion may arise equally from the implementation, if the program confronts the user with differences in the operations which related to the internal representation, but not to different user concepts.

Review Questions

- What is the difference between a collection of points interpreted as *1*-cell or as *2*-cells? What are the structural and what the behavioral differences?
- What is Jordan's curve theorem stating?
- Explain an algorithm to produce a collection of line segments that are not intersecting each other.

- What is the convex hull? To what operation is the complex hull dual?

Chapter 29 SUBDIVISIONS ARE PARTITIONS OF SPACE

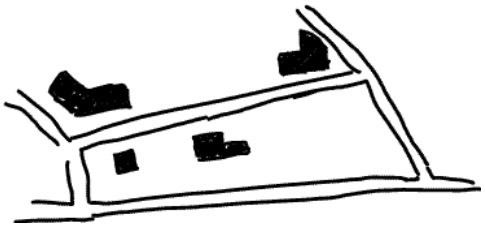
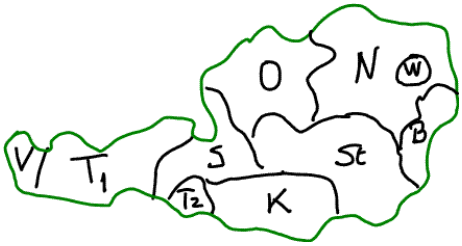


Figure 545: Buildings, streets and 'free' space

Figure 546: Austrian Länder



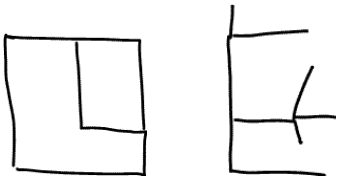


Figure 547: Example for subdivision and a graph which is not a subdivision

JEPD:
Jointly exhaustive and pairwise disjoint.

Representation of areas is in a GIS; isolated regions are sometimes of interest, but in general, geography is interested in the subdivision of space in regions, not only the buildings and the streets are, but also the 'free space' between them. This part will discuss such subdivisions of space, for which a prime example are cadastral parcels. They are spatial partitions, where every piece of land belongs to exactly one cell.

1. INTRODUCTION

Subdivisions of space that are constructed such that all the pieces exhaust (cover) all of the area and the pieces are not overlapping are common: political subdivisions are, for example, constructed this way and any political map showing the countries of Europe or the communes in a Bundesland has this structure (Figure 546).

Different approaches are possible to treat and represent subdivisions:

- to represent the partition by the lines that form it.
- as a graph, where we interpret the cycles in the graph as areas, called *faces*, or
- as a cell complex using the methodology of combinatorial topology.

In a GIS, the faces in the subdivision are associated with some thematic value, for example the cadastral identifier of the parcel, the owner, etc, or the soil type, the amount of rainfall annually, etc.

2. DEFINITION OF PARTITION

Let $P_I = \{A_i \mid i \in I\}$ be a set of non-empty subsets of A. The set P_I is called a partition of A if every element of A is in exactly one of the $A_{i(Gill\ 1976,\ 15)}$:

$\cup$ *over i,* $A_i = A$	*exhaustive*
A_i *intersection* $A_j = 0$ *whenever* $i \neq j$	*pairwise disjoint*

We call this property 'jointly exhaustive and pairwise disjoint' and abbreviate it to JEPD.

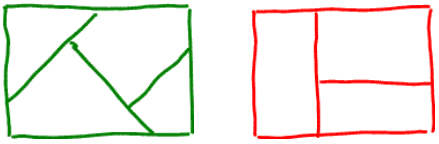


Figure 548: Two partitions

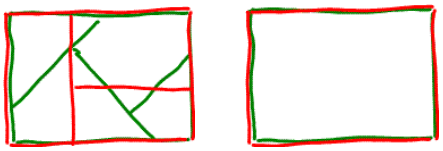


Figure 549 Their meet and their join

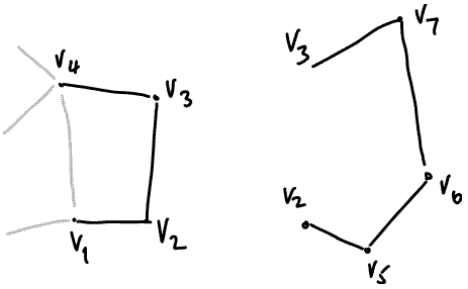
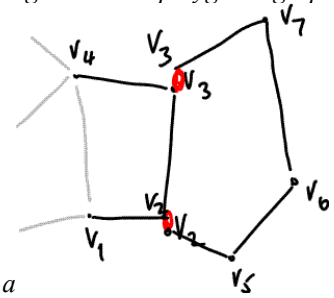


Figure 550: A polygonal graph and



a

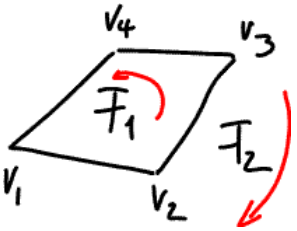


Figure 551: A polygonal graph and its completion. Note that face F_2 has negative orientation

Partitions are partially ordered by a relation≤. A partition P_I is finer than P_J if every $A_i \in P_I$ is a subset of some $A_j \in PJ$. We say P_I is a refinement of P_J and write $P_I \leq P_J$. Partitions form a lattice with this ordering. The operations *meet* and *join* are intuitively understood spatially: the meet is the partition with all the boundary lines and the join is the partition that retains only the common boundary lines (Figure 549).

3. POLYGONAL GRAPH

The set of boundaries in a subdivision (spatial partition) form a graph, which is called a polygonal graph. This graph has some properties, which we will describe recursively (followingGill 1976, 391)

Figure 580-11:—recursive definition of boundary graph

A polygonal graph is a connected, planar graph (condition that no edges are not crossing in *G*). It subdivides the a *2d* plane (or the sphere, the projective plane) into regions (called faces).

> A polygonal cycle is a closed, minimal path (see xx). A graph consisting of a single polygonal cycle is a polygon.

(Basis of recursion): A polygon is a polygonal graph.
(Induction step): Let G = (V, E) be a polygonal graph.
Let a = v_i, v_m, v_{m+1}, … v_n, v_j

be a proper path of length $l > 1$ which does not cross over *G* and where v_i and v_j in *V* and v_m…. v_n not in *V*. Then the graph G' = (V', E') with

$V' = V \cup \{ v_m \ldots v_n \}$
$E' = E \cup a$

is a polygonal graph. This recursive construction does not allow holes in a face, but allows cycles of length 2 (zweieck) or even length 1 (loops).

Each face is bounded by a minimal cycle. The maximal polygonal cycle of the polygonal graph is its outside boundary. It is convenient to consider this outside as an additional face; infinite face, outer void or similar names are customary. This completes the Euclidean plane to the projective plane (Figure 551). This outer face has the opposite orientation of the other faces, because the projective plane cannot be oriented consistently; all operations must avoid touching this 'outer face'.

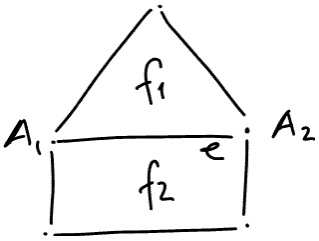


Figure 552: A polygonal graph with two faces

Faces are the same as *2*-cells in a complex; two faces are adjacent if they have a common boundary (in Figure 552: f_1 and f_2 are adjacent). A face is incident with the edge that is its boundary (in Figure 552 f_1 is incident with e). We may say an edge bounds a face or a face is bounded by edges.

3.1 BOUNDARY GRAPH MUST BE CONNECTED

The recursive definition of the subdivision enforces that the polygonal graph remains connected. This does not allow isolated holes (Figure 553).

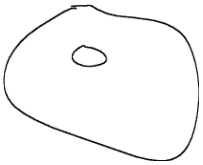


Figure 553: Isolated hole - not a subdivision!

3.2 RESTRICTIONS ON SUBDIVISIONS

What restrictions should be enforced? What assumptions can be built into the representation? Restrictions built into the representation makes it impossible for an application to construct objects that violate the rules, even if the restriction is not justified by the application. For example, spatial subdivisions have holes (Figure 546). Much of the discussion about optimal data structures for GIS center around questions what assumptions are built into a data structure, what application situations are excluded by these restrictions and how to circumvent them.

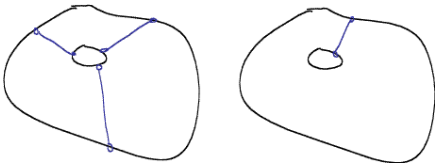


Figure 554: Two subdivisions constructed from cells

I have not seen a simple algebra for the maintenance of a polygonal graph allowing holes, i.e., a graph which is not connected. This makes polygonal graphs and *2*-cells not attractive as a method to implement directly subdivisions. The alternative is to subdivide the faces into *2*-cells (which are topologically equivalent to a sphere and have no holes) or into *2*-simplices (Figure 554). In either case, the operations on the subdivisions are translated into operations on aggregates.

4. EULER OPERATIONS ON SUBDIVION

The Euler polyhedron formula is invariant for a subdivision. Operations changing the subdivision must preserve this invariant; the elementary operations to change the polygonal graph are therefore called *Euler operations*.

4.1 EULER'S POLYHEDRON FORMULA

For polygonal graphs a relation between the number of nodes (*n*), edges (*e*) and faces (*f*) for a simply connected graph topologically equivalent to a cell is:

$N - E + F = 1$ *Euler's formula for a disk*

This formula is valid for simple connected graphs in *2d* (not counting the outer face). For example, the Figure 552 has *5* nodes, *6* edges, and *2* faces: *5 – 6 + 2 = 1*. The formula is proven by induction along the lines of the recursive construction of the polygonal graph above. The Euler polyhedral formula is most used for subdivision of the surface of a sphere or the projective plane, where the formula is

$N - E + F = 2$ *Euler's formula for a sphere*

because, the outer (remainder) face counts as well.

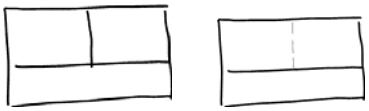


Figure 555: Glue

Figure 556: Split

4.2 GLUE AND CUT

Operations to change a polygonal graph into another polygonal graph must leave the Euler formula invariant. Inserting edges must divide faces; removing edges must merge faces. Traditionally the operations are called merge or **glue (**Figure 555**)**, divide, **split**, or cut **(**Figure 556**)**. Euler operations are the minimal steps that differentiate two partitions and all partitions can be constructed by a finite number of splits; any two partitions can be transformed in any other by a finite number of glues and splits.

Any partition can be constructed by splits.
Any partition can be transformed in any other by glue and split operations.

4.3 ORDER RELATION OF SUBDIVISION

A single cut operation makes a subdivision an elementary step finer than the original one; a single glue operation makes it an elementary step coarser.

There is a connection between a partition of a thematic space and the induced spatial subdivision. Consider the political subdivsion of space in countries, states, districts etc. (figure earlier). Different levels of this partitions lead to different levels of spatial subdivisions: the map of Europe as countries, the map of Austria with their Länder (states) (figure earlier xx), etc. It is possible, to subdivide one entry in the thematic partition further (e.g. Austria) but leave the other countries in the map of Europe the same (Figure 557). In general, any partition of thematic space produces a partition of geographic space. This mapping between thematic space and geographic space is order preserving: The partition of attribute space can be finer or coarser, and correspondingly is the subdivision of space finer or coarser(Frank, Volta et al. 1997).

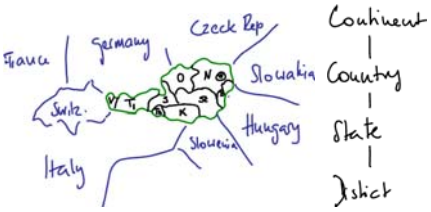


Figure 557Subdivision of Europe with Countries and sates for Austria

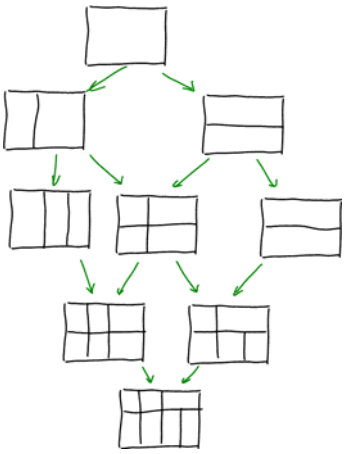


Figure 558: Refinement of subdivisions by cut

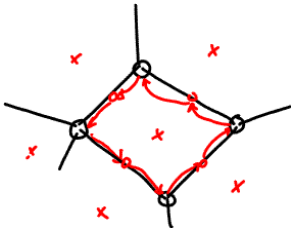


Figure 559: Closed path around face

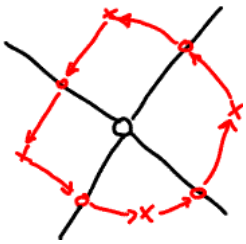


Figure 560: Close path around node

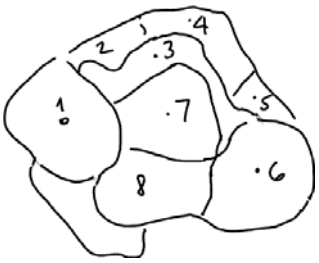


Figure 561: Spaghetti and meatballs resulting from digitizing

5. Invariants Used for Testing Partitions

The U.S. Bureau of the Census used in the early 1960s computers to prepare the maps for all the enumeration districts. These 50'000 xx maps were produced for the investigators who visited all dwellings in the assigned area and counted the persons living there; to assure that the whole population is counted only once, these maps must be JEPD and it was necessary to check the graphs which resulted from digitizing the street network for correctness. Corbett in a classical contribution—perhaps the first application of topology to GIS—proposed two tests which check a graph to see if it forms a proper partition(Corbett 1975).

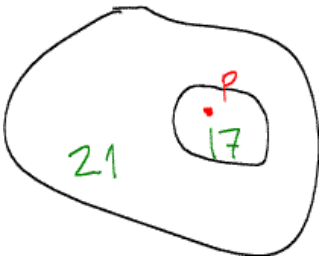
1. Closed path around an area: The cycle around an area must be closed, this relates to Kirchhofer's law, which says that the sum of the potential differences around a mesh in an electrical network is zero. This excludes isolated edges, missing edges, etc. but primarily inconsistencies in the polygonal graph structure (Figure 559).

2. Area closes around a point: The succession of face—lines—face must be closed around a point. This relates to the other law of Kirchhofer, that the sum of the inflow and outflow of a node must be zero (Figure 560).

6. Construction of Partition from Collection of Lines—Spaghetti and Meatballs

Manual digitizing of a boundary map resulted in a collection of lines, with arbitrary starting and ending points; instructions to the operator may result in some additional structure in the order with which the digitizing proceeded, but such hints were found to be unreliable. To collect information about the region, the operator digitized an arbitrary point inside a region and attaches the attribute information to this point (Figure 561). This follows the cartographic tradition of regions with labels (name of land, value of land, parcel number, etc.).

The result of digitizing boundary lines and labels for the regions without any particular order is in the jargon called 'spaghetti and meatballs' (after a popular dish in the USA). How to construct an algorithm to convert automatically such data into a subdivision? The code seems straightforward: compute all the line intersections and then construct the cells, following the edges around a node and around a face (invariants listed above).



Point p is inside of face 17 and 21—not a proper subdivision

At the end, identify the faces with a 'point in polygon' test for each 'meatball'.

This does not work, if the inputs contains ‘holes’—which are frequent in choropleth maps, but also soils map, political subdivisions, etc., contain them. Closing such polygons does not work with a method following a polygon around a node or face. Identifying the inner face later with a point in polygon test results in two polygons. Digitizing instructions often request that the 'island polygon' is connected by an arbitrary link to the outer boundary of the enclosing polygon (Figure 554).

7. CONCLUSIONS

A subdivision has the properties of a partition, it is JEPD. The operations on subdivision are restricted to operations that maintain the invariants of the partition. The invariant is succinctly expressed in Euler's polyeder formula, which is maintained by the Euler operations *glue* and *split*. It is sometimes necessary to combine several steps from a consistent partition to a new consistent partition.

To maintain a subdivision, the Euler operations are sufficient (Figure 562). The operation that produces a finer partition are implemented as integration of a point or a line (*0*-cell or *1*-cell) with a *2*-complex (see chapter 25). This cuts the edge or the face in two. The split and glue operations require only the creation of a new aggregate and a note that this aggregate contains the two cells.

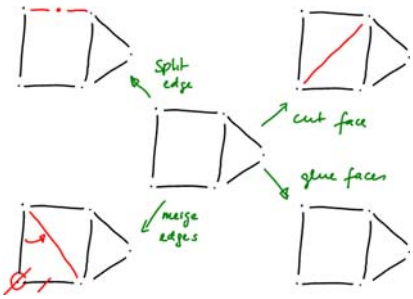


Figure 562: The two Euler operations on faces and the operations on edges

For practical purposes, an interface at a lower level manipulating directly the polygonal graph seems desirable. Operation could include:

- Add Node with Coord—to create a node with given coordinates and no connections;
- Delete node (with all connected edges);
- Connect nodes—creates a new edge between to existing nodes;
- Delete edge;
- Split edge with a node—insert a node into an edge;
- Delete node from two connecting edges.

These operations would not maintain the invariants of a polygonal graph and the user has to check at the end of a transaction consisting of several changes that the result is consistent.

REVIEW QUESTIONS

- Why a subdivision is called topological data structure?
- What is a ‘winged edge’ structure; why this name?
- Manual digitizing of partitions produces ‘spaghetti and meatballs’—what is meant with this jargon expression?
- What does JEPD mean?
- What are the Euler operations?
- Why does glue and split leave the Euler formula invariant?
- What are the tests to check the consistency of partition?
- Why does the recursive construction of the subdivision not permit the inclusion of isolated holes?
- What is the dual graph to a given graph?
- What is a complex? What is included, excluded?

Chapter 30 GRAPH DUALITY FOR TOPOLOGICAL DATA STRUCTURES

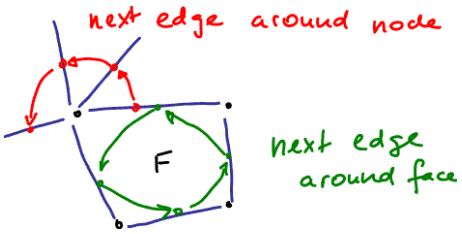


Figure 563: Operations to find the next edge around a node and the next node around a face

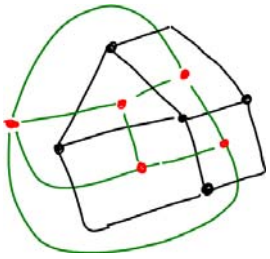


Figure 564: Strict duality

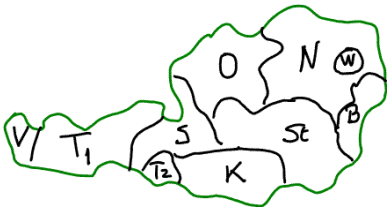


Figure 565: Austrian Länder

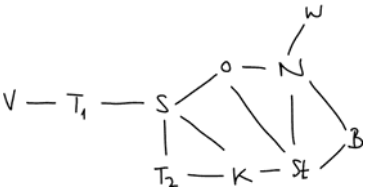


Figure 566: Dual of Figure 565

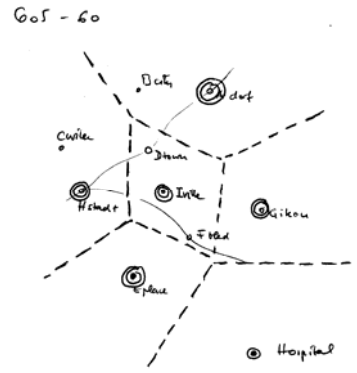


Figure 567: The areas of influence for a number of places

The interpretation of a graph as delimiting faces creates a need to be able to navigate around faces and nodes (Figure 563). This operation is used to ascertain that the graph remains planar and describes a subdivision (chapter 25 and 27). The same operation to visit all nodes around a face is necessary to compute the area of the face, to test if a point is inside a face, etc. The similarity of these two operations, shown in the previous chapter, is here explained by graph duality.

To a given *primal* graph a *dual* graph can be constructed, such that every face in the primal graph corresponds to a node in the dual graph and vice versa. Edges are mapped to edges that are crossing the primal edges (Figure 564). The dual graph gives the neighborhood relations between the faces (Figure 565, Figure 566).

Duality helps in GIS to determine areas of influence around point objects. For example: what is the area served by a hospital (Figure 567)? The assumption is that a service point (e.g. a hotel) services all points which are closer to this service point than to any other. It is given by the Voronoï diagram (also called Thiessen polygon) around the given points. The construction of the Voronoï diagram can be done directly but it is more convenient to use duality and construct first the dual of the Voronoï diagram that is the Delaunay triangulation.

The Delaunay triangulation is optimal in the sense that all the triangles are as similar as possible to isocycle triangles. There are many possible triangulations for a given set of points, but only one Delaunay triangulation. It is unique and this alone is sometimes useful.

1. GRAPH DUALITY

We have used duality before, most productive in projective geometry where we had a duality between points and hyperplanes (chapter 19). In subdivisions, there is a duality between the faces and the nodes. An edge has two adjacent nodes and two adjacent faces, which hints to graph duality.

Graph Duality:
Node ←→ Face
Edge ← → Edge (crossing)

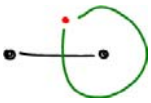


Figure 568: A graph with one edge and its dual

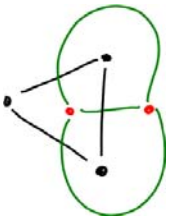


Figure 569: A graph with two faces and its dual

Duality for graphs: every correct sentence about a graph is correct if node and face are systematically interchanged.

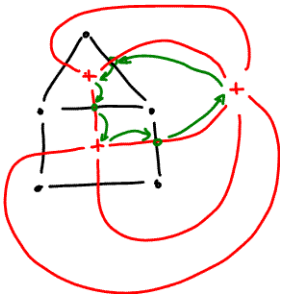


Figure 570: Cycle around a node

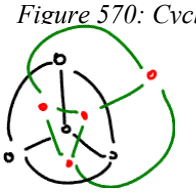


Figure 571: A self dual graph

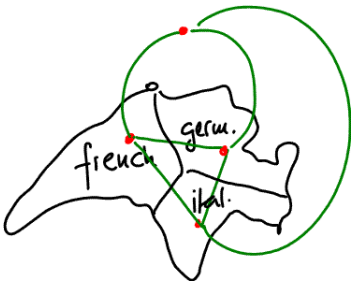


Figure 572: A geographically interesting example for self dual graph: Switzerland and the three language regions (French, German and Italian speaking)

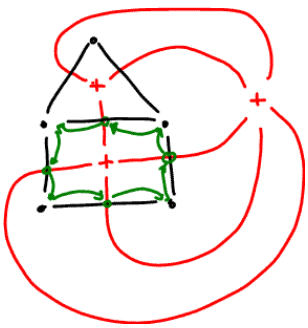


Figure 573: Dual: cycle around a face

The primal graph shows as edges the node adjacency relation; the dual graph shows face adjacency relation. This can be interpreted in applications: For example, in Figure 566 we can see which regions of Austria are neighbors and which ones are not (the connections to the outer face are left out).

The dual graph of a graph is constructed as follows: Represent each face as a node in the dual graph (including the infinite face). Replace each edge with an edge crossing. Figure 568 and Figure 569 give two simple graphs and their duals.

The dual of a planar graph is always planar. Duality for connected graphs is its own inverse:

dual . dual = id.

The dual of a cell is different from the element; duality separates the two graphs in two sets of elements, each consisting of edges, nodes, and faces, where duality maps between faces and nodes and maps edges to edges. There are graphs that are self-dual, i.e., the graph and its dual are identical (Figure 571), but real-world regions with this property are seldom (Figure 572).

The degree of each dual node representing a face is equal to the number of edges bounding this face, and, by duality: the number of edges of a dual face is equal to the degree of the node. The dual of a triangulation (all faces have three edges) results in a graph where all nodes have degree 3, but not in a triangulation (Figure 582)! The two consistency tests (in chapter 29) are in fact only a single test, duplicated by duality (Figure 570, Figure 563, Figure 573).

If we represent the graph as relations between nodes, edges and faces, we can see immediately that the dual graph is available when we interpret the node-edge relation as the face-edge relation and vice versa. The only difference between nodes and faces is that the faces have no coordinates; it is useful to select for the faces the mean of the coordinates of the 0-skeleton (corners of the triangle). This point is inside the triangle and is the center of gravity. No such simple rule exists for arbitrary cells.

1.1 WHY NO HOLES? WHY NO ISOLATED NODES?

Duality helps to understand the difficulty with isolated holes (previous chapter) and see that isolated nodes create similar

No holes, no isolated edges guarantee that dual.dual = id

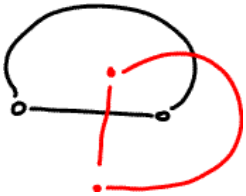


Figure 575: Dual of the dual of Figure 574

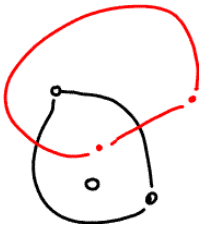


Figure 574: Graph with isolated node and dual

Figure 576: A graph with a hole

Figure 579: Edge and dual edge

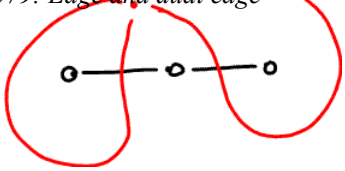


Figure 577: The dual graph to the dual of Figure 576

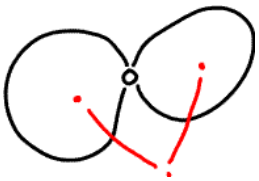
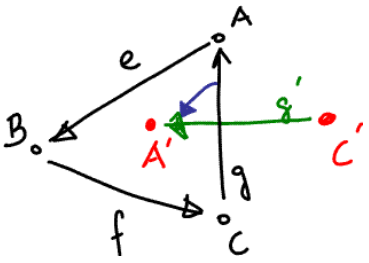


Figure 578: The dual of the dual



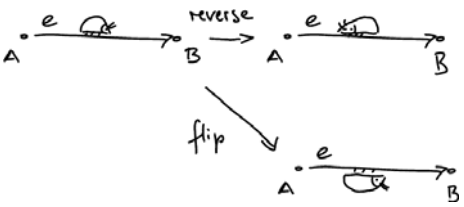


Figure 580: The bug giving direction and orientation

difficulties. The dual of the dual of a graph, which is not connected, is not the same graph.

There is an asymmetry between primal and dual graph. A graph with a hole has a dual with an isolated edge, but the dual of the graph with the isolated edge does not give a hole. Figure 576 shows two applications of the operation dual: the first graph has a whole and its dual is shown. The dual of this figure is a connected graph with three faces (2 inner ones, one exterior), which are not inside each other (Figure 577). The next figure shows that the dual of this graph is again the second graph (Figure 578). Duality works only for connected graphs!

A similar argument excludes isolated nodes. They do not show in the dual graph and dual would not be a self-inverse (Figure 574, Figure 575).

1.2 Duality and Orientation

The figures above show nodes and edges, but do not consider orientation. Unfortunately, orientation is reversed by duality. In Figure 579 the dual of edge *g* is *g'*. It is the result of turning the edge *g* by a function *d* turning one quarter in positive direction. Applying the same operation *q* to *g'* gives not *g*, but the *reverse g*. Applying dual to *g'* must give *g* (*dual. dual = id*), but left applied to *e'* gives (reverse *e*).

Duality is only achievable with a trick: look at dual part of Figure 579 from the back of the page. Then turning *e'* anticlockwise (positive) gives *e*, as desired. The strict dual graph is the graph constructed by replacing faces with nodes and vice versa and turning edges, but looked at from the 'other side', the flipped side (alternatively, one can define positive orientation differently for primal and dual graph). A hint to this asymmetry in duality was seen, when following the next edge around a node (in anticlockwise direction) and uses the same pointers to circle around a face clockwise (chapter xx).

To achieve operations which use duality extensively, we have to separate direction and orientation of an edge. The orientation of an edge determines what is left and what is the right face bounded by this edge (above, orientation was fixed given the direction). Orientation and direction are two properties and independent from each other; one can “picture the orientation and direction of an edge as a small bug sitting on the surface over the midpoint the edge *e* and facing along it. The

operation *reverse e* corresponds to the bug making a half turn on the same spot, and *flip e* corresponds to the bug hanging upside down from the other side of the surface, but still at the same point of the edge and facing the same way" (Guibas and Stolfi 1982, 80)(Figure 580).

It is to understand this limitation of dual graphs in order not to expect from them properties that they cannot have.

2. VORONOÏ DIAGRAMS GIVE 'AREA OF INTEREST'

Consider a number of service points—for example shopping centers or hospitals—and delimit the area served by each point. This concept of '*service area*' is an often used concept, useful in many applications.

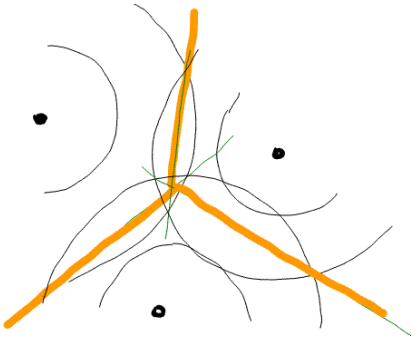


Figure 581: Service points and distance to them

The application concept of 'area of interest' 'area of influence' or 'service area' must be translated into a formal definition that captures the relevant aspects of the concept. Start with a set of service points, which we will call nodes. Assuming that the space is isotropic and any point will prefer service by the node that is closest. This gives a definition of service area, as the region of all points that are closer to one service point than to any other service point. Each point of space is serviced by the service point closest, or, every service point provides service to all points that are nearer to this point than to any other point. This gives areas around each point as shown in Figure 581.

The construction of a Voronoï diagram starts with the middle points between any two points (M, M', M" in Figure). These points must be on the boundary between two regions. All points on the bisectors between the two points are potentially also part of the boundary between two service areas. Bisectors of three service points close have a single intersection point. This gives the boundaries of the Voronoï regions. If many service points are given, then the manual determination of which intersection points are meaningful may be confusing, but is not a principal problem.

Figure 582: A Voronoï Diagram and the dual Delaunay triangulation

Combining such a Voronoï diagram with population density gives us an idea how many persons are serviced by each point—the assumption that people go to the next service point is a best first guess. This overlay operation of population density (or a modified value, population in a certain age group) with Voronoï regions is one of the basic operations in "business geography" which supports spatial marketing decisions [ref].

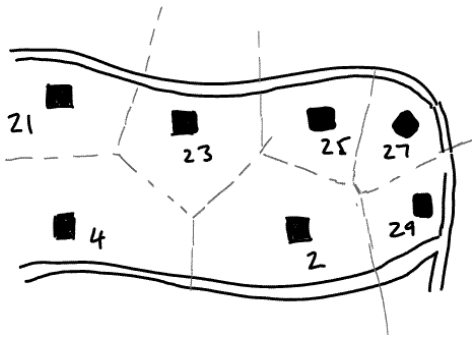


Figure 583: Which buildings are neighbors?

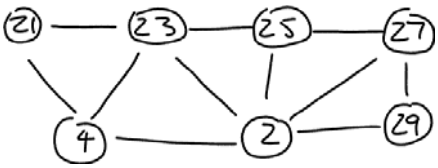


Figure 584: The dual graph to the Voronoï diagram in Figure 583

3. VORONOÏ DIAGRAMS STRUCTURE EMPTY SPACE

The Voronoï diagram is not only useful for the determination of service areas, but is also used to structure the empty space between the observed features. Maps show salient features, but most of the space is left in the background color. Jones has suggested that the Voronoï diagram for the features shown assign to each feature some of the open space(Jones, Bundy et al. 1995). This is then later useful to determine if two objects are neighbors, for example two buildings, which are not touching are considered neighbors if their Voronoï area of influence are touching (Figure 583).

In the figure, some buildings along two streets are shown. The neighborhood relation defined through the Voronoï diagram makes 2 a neighbor or 27, but not 25 a neighbor of 29, because their distance is, compared to the distances to other buildings too large. The dual of this graph gives the neighborhood relation (Figure 584).

4. BARRIERS AND NON-POINT SOURCES

A service point may not reach all the location just based on proximity—hard boundaries in the terrain may make this impossible. For example a river may make it impossible to reach the nearest distance service point. Space is not isotropic in this case and the Voronoï diagram must include these boundaries.

A different complication is introduced by services that are not points but lines or regions. For services given as lines—e.g., a road that can be accessed any place—the boundaries of the Voronoï diagram are parabola. They are the geometric locus of all points having the same distance to a point and a line!

5. DELAUNAY TRIANGULATION IS THE DUAL OF A VORONOÏ DIAGRAM

The primal nodes are the given service points. The Voronoï diagram has the special property that three boundary segments meet in a single point (Figure 581). These intersection points of the boundaries will be the dual nodes forming the graph of the Voronoï diagram. There are three primary edges connecting three service points around each of the dual nodes. The dual is a triangulation and the primal and dual edges cross at right angles (Figure 585).

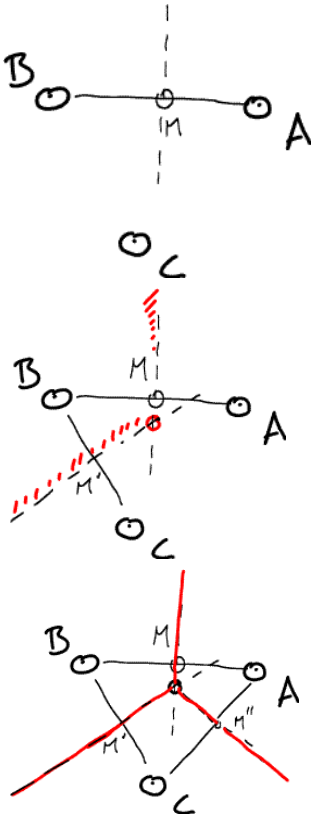


Figure 585: Construction of the Voronoï diagram

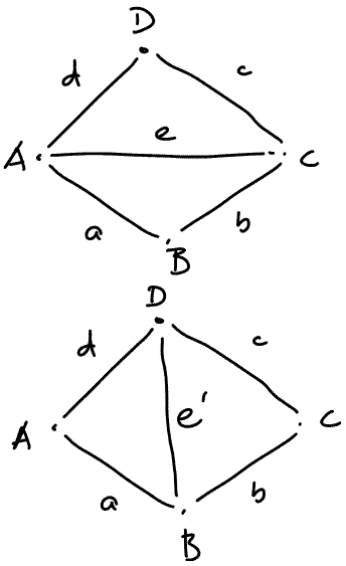


Figure 586: The originally constructed triangulation and the switched one

$$\text{inCircle} = \det \begin{vmatrix} x_A & y_A & x_A^2+y_A^2 & 1 \\ x_B & y_B & x_B^2+y_B^2 & 1 \\ x_C & y_C & x_C^2+y_C^2 & 1 \\ x_D & y_D & x_D^2+y_D^2 & 1 \end{vmatrix} > 0$$

We have assumed here that the service points are in general position and not any 4 of them lie on a circle. Only then all the nodes in the Voronoï diagram have degree 3 and the dual—which is called Delaunay triangulation—is a triangulation (Figure 582). This triangulation is well-determined for this non-degenerated case: for any set of points there is exactly one Delaunay triangulation.

5.1 Construction of Voronoi Diagram and Delaunay Triangulation

The construction of the Voronoï diagram is somewhat complicated and it is often easier to find the Delaunay triangulation and then to construct the dual graph. An incremental algorithm to construct a Delaunay triangulation inserts point by point in a triangulation. In addition to the methods introduced in chapter 25 for the construction of a simplicial complex (i.e. a triangulation), we check after the creation of a new triangle, if this triangle has the properties of a Delaunay triangulation or the switched triangulation of the four points would be better (Figure 586). To determine whether ABC and ACD are the better triangulation than ABD and BCD we use the incircle test (see chapter 9).

The incircle test checks whether a point is inside the circle defined by the three other points. A triangulation is Delaunay if all its edges pass the circle test.(Guibas and Stolfi 1982). The test is for the Figure 586 (upper) incircle (ABCD) > 0 and for the lower figure incircle (ABDC) > 0 (note that incircle ABCD = - incircle (ABDC); it is one permutation of the matrix from which the determinant is taken (Figure 587).

5.2 Details of Incircle Test (Review):

Given three points ABC, not collinear, *incircle (A,B,C,D)* is true, if A B C defines in clockwise order a triangle and the point D is inside the circumcircle of this triangle (Figure 587). This is equivalent to test

Angle ABC + Angle CDA < angle BCD + angle DAB.

The test can be written as a determinant (for details see Guibas and Stolfi(Guibas and Stolfi 1987):

Reversing the order of the points gives the negation of the predicate (i.e., true becomes false, false becomes true), as does transposing any adjacent pair.

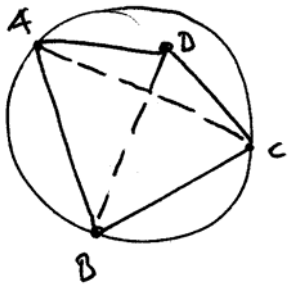


Figure 587: D is inside the circle defined by ABC

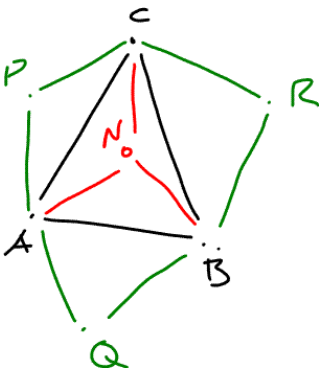


Figure 588: The point N is just inserted

5.3 USE OF THE TEST

The new edges that are created when inserting a point into a triangle are Delaunay (see Lemma 10.1(Guibas and Stolfi 1982)) and only the previously drawn edges in the triangle or quadrilateral are suspect (i.e., the edges AB BC CA, or AB BC CD DA in Figure 588). These must be tested by the incircle test and swapped if necessary. If all suspect edges are tested, the triangulation is Delaunay and the next point can be inserted.

6. ALGEBRA TO MAINTAIN A *2D* MANIFOLD

It instructive to study the algebra proposed by Guibas and Stolfi (1982) to maintain a *2d* manifold. The general algebra maintains *2d* manifolds. The restricted case for *2d* subdivisions can be deduced. Their algebra does not allow holes or isolated points.

The method proposed by Guibas and Stolfi is treating the primal and the dual graph at the same time. The two graphs together give a triangulation of space (Figure 589), known as the barycentric subdivision(Henle 1994, 130).

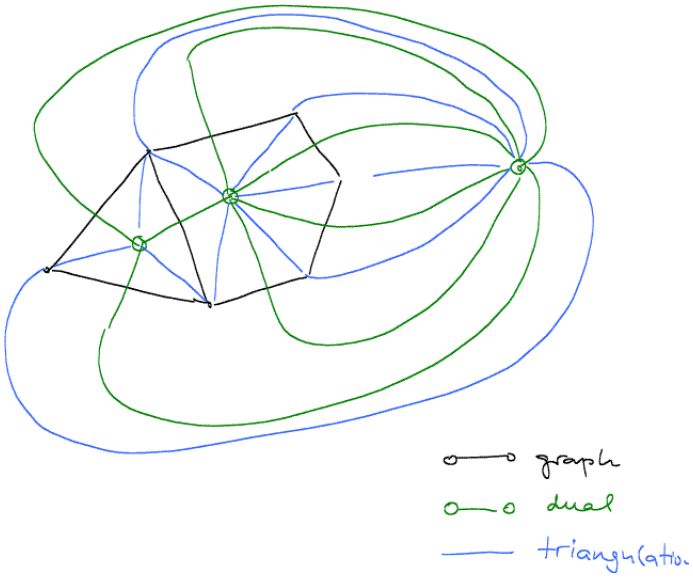


Figure 589: Priaml and Dual Graph give triangulation

6.1 MANIFOLD (GERMAN MANNIGFALTIGKEIT)

The algebra constructs and maintains a *2d* manifold. This is a surface which is locally everywhere *2d*. A manifold of *2*-dimensions is a topological space, where the neighborhood for every point is equivalent to a disk. This includes surfaces that are

not planes or not orientable, for example the Moebious strip, Klein's bottle, etc. The definition for a subdivision of a manifold is(Guibas and Stolfi 1982, 77):

A subdivision of a manifold *M* is a partition *S* of *M* into three finite collections of disjoint parts, the vertices, the edges and the faces, with the following properties:

S1. Every vertex is a point of *M*

S2. Every edge is a line of *M* (A line is subspace of *M* homeomorphic to the open interval (0,1))

S3. Every face is a disk of *M* (A disk is a subspace of *M* homeomorphic to the open circle with unit radius)

S4. The boundary of every face is a closed path of edges and vertices.

Two subdivisions *S* and *S'* on two manifolds *M* and *M'* are equivalent, if a homeomorphism of *M* onto *M'* gives an isomorphism from *S* to *S'* that maps each element of *S* onto an element of *S'*. The converse is not always true: Not for every isomorphism between two graphs *S* and *S'* exist a homeomorphism between the manifolds. A topological property of a subdivision is a property that is invariant under equivalence(Guibas and Stolfi 1982, 79).

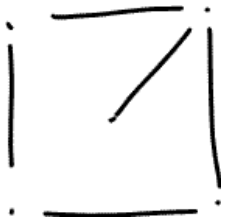


Figure 590: Manifold, but not polygonal graph

The difficulty is to define a representation such that it represents all the valid subdivisions, and not more and not less. The *2d* manifold is broader than what is necessary to represent subdivisions; our immediate purpose is the maintenance of a planar, orientable surface. A *2d* manifold is not necessarily planar or orientable. A manifold admits also edges that are not boundaries, i.e., which have the same surface on both sides (Figure 590). For the purposes of maintaining a 2d subdivision, which is a special case, a number of simplifications can be introduced.

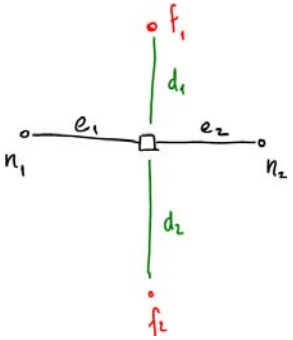


Figure 591: Combine half-edges for primal and dual graph gives quad-edge

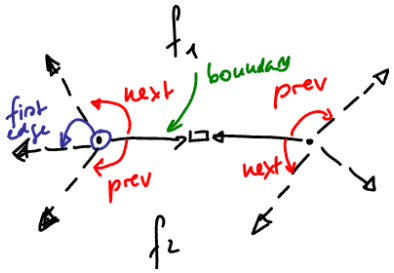


Figure 592: The quad-edge (g, l, h, m), where g,h are primal half edges and l,m are dual half edges

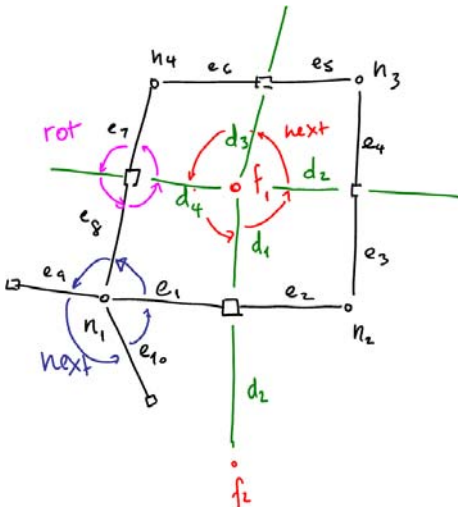


Figure 593: The functions rot and next

6.2 ALGEBRA WITH QUAD EDGES

An edge and its reverse are represented as half-edges (see chapter 27xx). If we add the two half-edges from the dual graph to the half-edges from the primal graph, we get a quad-edge (Figure 591). The algebra with quad edges represents each edge by four parts: $e_1$and e_2 are the primal quad-edges (e_1 is the reverse of e_2) and d_1 and d_2 are the dual quad-edges. The four quad-edges are connected by an orbit. The origins of e_1 and e_2 are the nodes n_1 and n_2, the origins of the quad-edges are the faces f_1 and f_2. Around the nodes and faces are orbits for to find the next quad edge (i.e., in constant time, following a pointer); the next function around a node gives the emanating quad-edges; the next function around a face gives the dual edges for the edges around the face.

The function *rot* gives the next quad edge, whereas the function next gives either the next quad edge around a node or the next dual quad-edge around the face. Note, that rot is not dual: $rot \,.\, rot \neq id$. These operations can be mapped to database relations (or pointers) and are fast. The advantage of quad-edges is that no backwards pointers are necessary. For example, finding the previous edge around a node ($next^{-1}$) is going from the half edge (say e_8) to the quad edge d_4 by *rot*; then going to d_1 by *next* around f_1 and then to e_1 by *rot*. This gives $next^{-1}(e_8) = rot\ (next\ (rot\ (e_8)) = e_1$; generally: $next^{-1} = rot \,.\, next \,.\, rot$)

6.3 ASSESSMENT

The quad-edge algebra was the first provably correct and efficient set of operations on a subdivision. The approach her is simplified for orientable surfaces and we must carefully restrict operations to the orientable part of the projective plane. The method does not directly deal with holes and should be combined with simplicial complexes (but then other simplifications may apply).

7. TOPOLOGICAL DATA STRUCTURES

In the GIS industry the term topological data structure (or short topology) refers to a representation of subdivisions, where the relations between node and edge (like a graph) and edge and face are maintained. This is essentially a graph and its dual, merged into a single structure.

Many proposals for data structures to represent the topology of a subdivision exist. They can be summarized in diagrams that indicate the elements that are stored, face, edge (or half-edge or quad-edge) and node. The arrows represent functions that lead from one to an adjacent element directly in constant time.

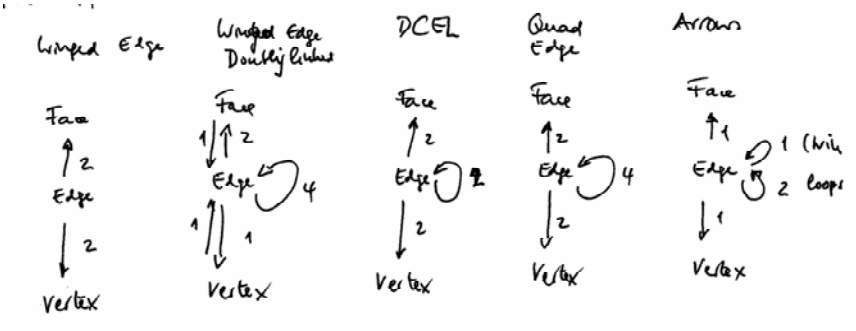


Figure 594: Different representations of subdivisions

Figure 594 shows a comparison of different proposals for data structures. Relations that are not shown as arrows are in this data structure combined from other relations—following an orbit around a face or a node—and take more time. One can see that the quad edge is one that has not more pointers than the most efficient other ones, but gives at the same time the dual graph.

Most often used is the 'winged edge structure'. Attractive is also the representations by arrows, which is essentially half-edge plus a pointer to the face.

7.1 WINGED EDGE STRUCTURE

The idea that a partition is a graph—to represent the edges and their adjacency with points—and the dual graph—to represent the edges and their adjacency with the faces—follows an often used data structure to represent partitions (Figure 595):

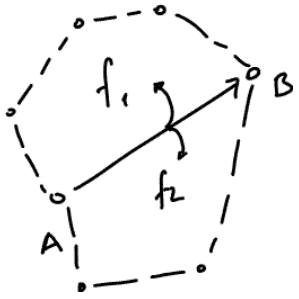


Figure 595: The edge AB is linked to A and B and to the two faces f_1 and f_2

This data structure has the advantage that each element has a fixed number of components. An alternative, where areas are represented by a list of the edges or a list of the boundary points would be much less convenient to deal with. The partition is represented by four functions

```
startNode, endNode :: e -> n
leftFace, rigthFace :: e -> f,
```

which are all proper functions with a single result (accepting the infinite face as a proper face). The disadvantage is that following the edges around a node or around a face is difficult and requires searching in the list of edges and requires the computation of angles and sorting the edges leaving in a node (Figure 596).

Figure 596: An example of a winged edge representation as tables

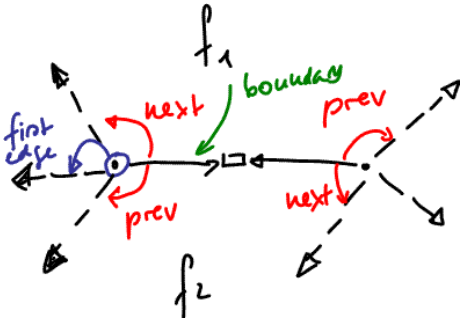
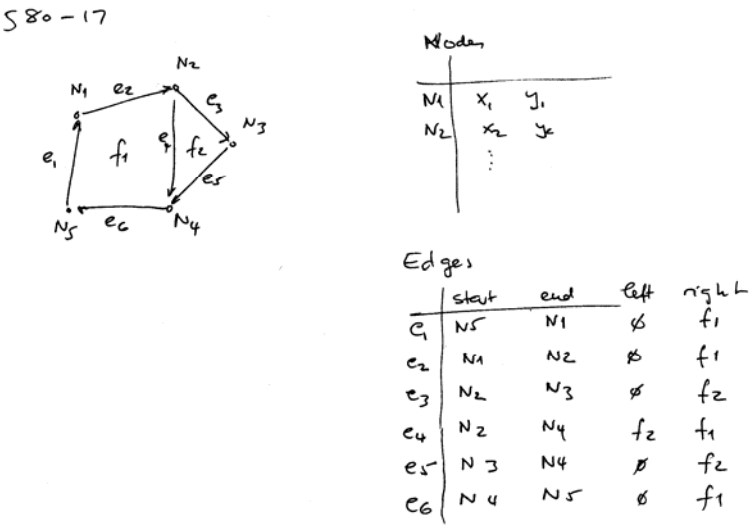


Figure 597: The half edges with additional pointer from face to an edge and from node to an edge



7.2 AN ALTERNATIVE TO QUAD EDGES: HALF-EDGES—ARROWS

In this representation, the edges are split in two ‘half-edges’ or arrows that emanate from an origin (the start node) and have a twin, which has as an origin the end node of the boundary edge. There are links leading from one half-edge starting at a node to the next around the node. These links together with the links between the twin half-edges make it easy to follow around a face (Figure 597).

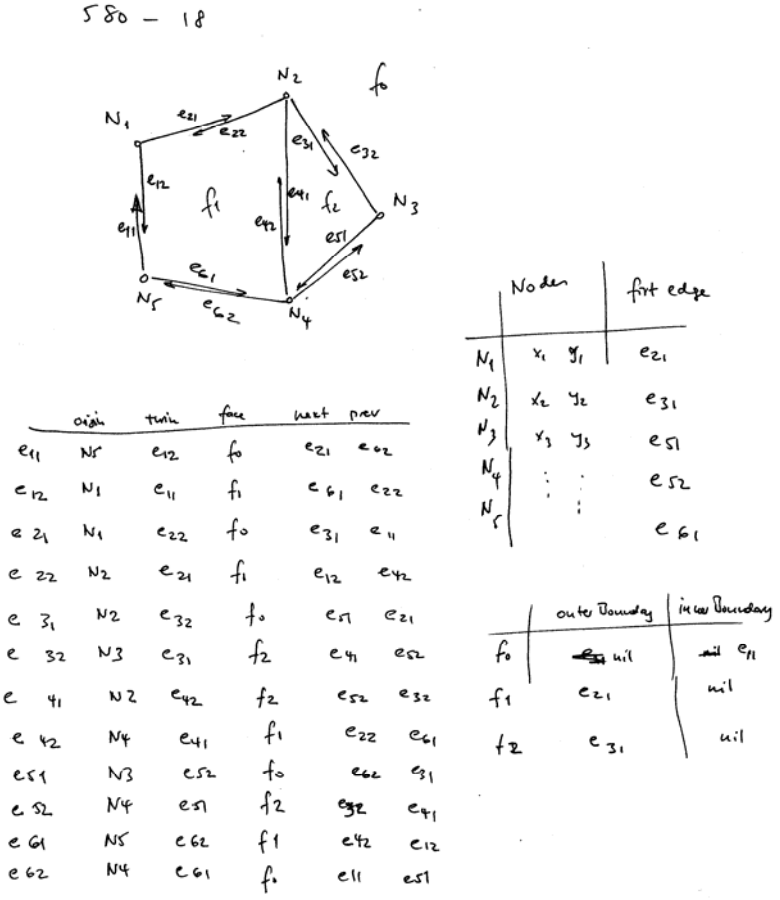


Figure 598: An example data structure

This data structure represents the functions:

```
firstEdge:: n -> e
origin :: e -> n
twin :: e -> e
face :: e -> f
next :: e -> e
prev :: e -> e
outerBoundary:: f -> Maybe e
innerBoundary :: f -> Maybe e.
```

This data structure is more voluminous than winged edge. Some of the relations that were immediate are now indirect. An example is

startNode = origin, but *endNode = origin.twin.*

To get all the edges around a face if one follows the *next* function. To go around a node, one follows *twin.next;* in both cases checking for the end of the loop by comparing with the element one has started with.

7.3 CRITIQUE:

These two descriptions of data structures lack definitions for the operations. It is difficult to program operations that guarantee that they remain consistent. It is also not immediate, what special cases of subdivisions are included or excluded.

8. SUMMARY

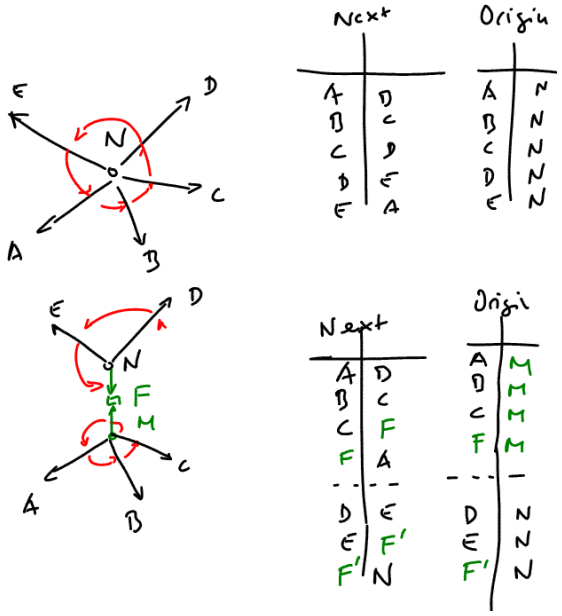
Duality links a graph separating the faces to the graph describing neighborhoods between faces. The Voronoï diagram and the Delaunay triangulation are dual to each other. They connect points, boundaries and areas in a way which is meaningful to many applications.

The algebra presented by Guibas and Stolfi is a formal approach to a long-standing problem, namely an algebra with a representation and operations for spatial subdivisions. The method maintains a *2d* manifold consistent. It has some disadvantages for GIS applications:

- It can handle non-orientable surfaces (like the sphere or the projective plane) but at a cost—the operation flip and the representation of its state.
- If it is restricted to an orientable subspace (for example the state plane coordinates for a country) then coding must ascertain that the 'inconsistencies at the boundary' are never encountered. There is potential for errors during execution.

The algorithm is short and effective, but it does only manage the orbits around the nodes and faces, it does not maintain the

relations between edges and nodes and faces, because the orbits are the representation of face and edge and are duplicated with the *origin* pointer.



The method is overly general and perhaps too general to treat the special case we are interested in. The management of simplicial complexes seems less demanding.

Review Questions

- What is a Voronoï diagram? What does it show?
- What says *dual* . *dual* = *id*?
- What is the dual of the Voronoï diagram?
- What is special about the Delaunay triangulation compared to other triangulations?
- Why did we not construct a 'which face is this point in' function?
- How do you determine the service area of some service points?
- Where are the 'point-in-circle' routines used? Why did we define them earlier?
- Why is the edge algebra outlined not easy to use?
- What are the consistency constraints of the insert edge and insert node operations?
- What are the inverses of *insert edge* and *insert node*? Give a graphical example.
- For what applications will we use the *insert edge* and *insert node* operations?

PART ELEVEN TEMPORAL DATA FOR OBJECTS

In this last—and short—part, we return to temporal data. The functor *changing*, whichwe have used to extend operations from static and local functions to time series in part 4, is applied to value describing the properties of objects. A moving object, for example a car, is nothing else than an object with a *changing location* and comparable to a town that has a *changing population.* The first chapter treats representation of objects moving in space, showing in detail how the functor changing is applied to what appears as single attribute value—here position. For a moving object, the position is a function of time. It can be observed and we obtain a time series not different from the observation of temperature in chapter 11 xx.. The application of this functor gives us the representation of the changing world.

A database reflects not the current world state but our knowledge of the world state. Our knowledge of what is the case in the world typically lags behind the changes in reality; similarly, the facts in the database most often describe what was the case earlier and may be changed already. The representation is influenced by limitations of our observations and the methods we use to classify the observations made; it may also contain gross errors and other inaccuracies. In administrative and legal procedures it is necessary to be able to demonstrate what was known at a given time in contradistinction to what was a fact at the same time. The database itself is then considered a changing object – namely our changing knowledge of the world. The same functor *changing* constructs the 'database time perspective' from a single spatio-temporal database.

In effect, applying the functor changing twice to a snapshot database gives a spatio-temporal database with both time perspectives: the changing world and the changing knowledge about the world. One can ask to types of questions: "Where was object X at time T" and "At time U, where did we believe that object X was at time T".

Chapter 31 MOVEMENT IN SPACE: CHANGING VECTORS

Movement of objects in space is important for humans and the representation of movement in GIS an opportunity to make GIS more useful. This chapter investigates in detail the representation of changing objects and the operations applicable. We separate the change of properties of the objects, including the location, shape, etc. and the creation and destruction of objects.

This chapter shows how the functor changing that was introduced in chapter 11 and used there to represent time series, is directly applicable to moving objects, e.g., the location of taxi cabs in a city or airplanes in the sky.

1. INTRODUCTION

Real object movement is complex and an Information System can only contain a simplified approximation. This chapter starts with the approximation of movement as piecewise linear and with a fixed speed (velocity *v*). The discussion is mostly treating uniform movement, but it shows also, how other movements with changing speed can be modeled with the same approach.

2. MOVING POINTS

Movement can be abstracted to the movement of point objects, or movement of the center of gravity of extended objects. Movement is controlled by a vector *v* indicating the speed of an object and the position p is the integration of this speed over time with the initial position p_0 (Figure 599). Initial position, velocity, and momentary position are all expressed as vectors; time is a scalar, as usual.

$p(t) = p_0 + v * t$

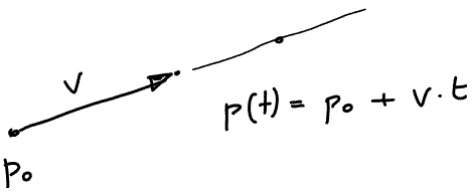


Figure 599: A moving point

Such a moving point is a point changing its position in time; this is a 'changing vector' and is the result of applying the functor 'changing' to a vector. The use of the functor changing converts a simple point (data type vector) in a changing point (exactly changing vector). Changing points, i.e., moving points, represent movement. The location of a moving vehicle is described as a function that yields a point for every moment in time.

If the object O_1, e.g., an airplane, is stored in the database not as an object with a fixed location, but as a moving object,

which is a changing vector, it is possible to ask questions like "give the location of O_1 at T_1" or "How far is O_1 from the airport now".

3. FUNCTOR CHANGING APPLIED TO VECTOR

The use of the functor changing applied to vectors gives the moving point. The ordinary operations addition and subtraction are lifted to work on changing vector. They can be used to calculate the distance between a moving object and a fixed location or to calculate position of an object that is moved relative to a moving reference frame (Figure 600). If $p_1(t)$ is the location of the object relative to the moving frame (e.g. a wagon) and the position of the frame is $p_2(t)$ then the position of the moving object relative to the outer reference frame is $(p_1 + p_2)(t)$.

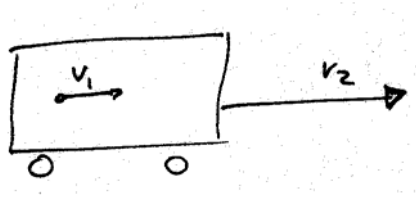


Figure 600: Object moved inside a moving object

4. DISTANCE BETWEEN MOVING OBJECTS

An interesting question is the distance between a moving object and a location or between two moving objects. Given a function to calculate the distance between two points p_1 and p_2; can this function be used to calculate the distance between two moving points? Lifting the function $dist\ (p_1, p_2)$ with the functor changing (*lift 2*) means that all the constant coordinates x and y of p_1 and p_2 are replaced by functions $x\ (p_1, t)$, $y\ (p_1, t)$, $x\ (p_2, t)$ and $y\ (p_2, t)$. The result is a formula to calculate the distance between moving points as a function of time. This is, of course, a synchronous application of calculations valid for a single time point, like the synchronous operations on time series in chapter 11.

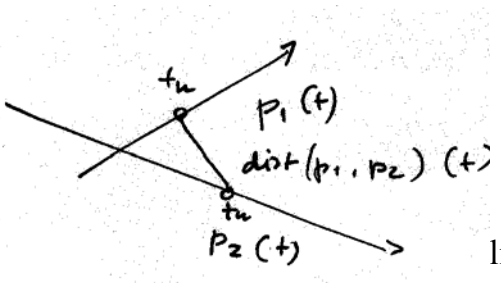


Figure 601: Distance between moving points

$$d\ (p, q) = \sqrt{(x_p - x_q)^2 + (y_p - y_q)^2}$$

$$d\ (p(t), q(t)) = \sqrt{(x_p(t) - x_q(t))^2 + (y_p(t) - y_q(t))^2}$$

If all the standard arithmetic functions are available in a lifted form to apply to changing values then lifting the distance function gives the desired function. A subtle conversion is necessary: a changing vector (of x and y) must be converted in a vector of two changing coordinate values: a changing vector and a vector of changing coordinates is semantically the same, syntactically different.

Terminology:
Speed a scalar describing the magnitude of the velocity vector
Velocity a vector describing speed and direction of movement

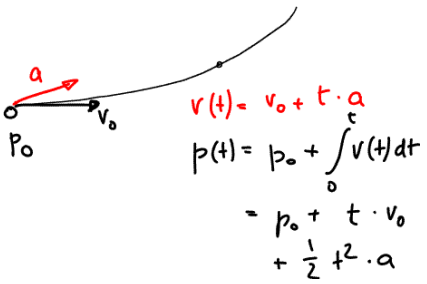


Figure 602: An accelerated movement

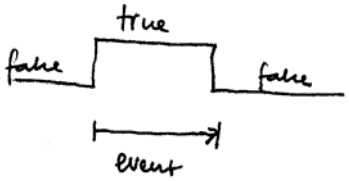


Figure 603: An event as a Boolean fluent

Note: event *in this definition is not a time point, but an interval.*

5. ACCELERATED MOVEMENTS

Point movements with constant acceleration can be described with the same functor: velocity is changing; a function of time and position is a function of the (changing) velocity. The trajectory of an accelerated movement is a straight line only if the initial velocity and the acceleration are parallel. If the acceleration is different in different directions and not parallel to the initial velocity, a curved trajectory results (Figure 602).

Lifting with the functor changing is not sufficient to obtain all results desired. Second order functions for integration of function values over time intervals are necessary. Software packages for treatment of formulae (Wolfram 1988) can do symbolic differentiation and integration for complex expressions and the same methods could be applied here. For realistic complex functions, numerical integration and differentiation is an option to calculate approximations(Wolfram 2002).

6. EVENTS

Allen—following the philosophic discussion in Hamblin—defined an event as the minimal interval over which a state holds. This is the same as a fluent of Boolean values, which is true within an interval and not outside (Figure 603).

This definition of event is not conforming to common usage of the word in the English language. Wordnet (Fellbaum 1998) gives 4 senses for the noun event:

> 1. event -- (something that happens at a given place and time)
> 2. event, case -- (a special set of circumstances; "in that event, the first possibility is excluded"; "it may rain in which case the picnic will be canceled")
> 3. event -- (a phenomenon located at a single point in space-time; the fundamental observational entity in relativity theory)
> 4. consequence, effect, outcome, result, event, issue, upshot -- (a phenomenon that follows and is caused by some previous phenomenon; "the magnetic effect was greater when the rod was lengthwise"; "his decision had depressing consequences for business"; "he acted very wise after the event")

The definition used by Allen is not the sense ordinary English gives to the term. It seems, unfortunately, to be the one generally used in Philosophy, AI and in discussion of temporal GI. Given that no better terminology is available, I will use it as well.

An event is defined as an interval of time (not a point in time) in which some property is uniform. This is parallel to the definition of objects as areas in space, which have uniform

properties (chapter 14). The property that is uniform and makes us see an object depends completely on the application; what we have in mind makes things objects or events—in other circumstances other events are identified.

Events and Objects are similar: temporal or spatial intervals (respective regions) with uniform properties.

A definition of events as closed intervals, where both start and end point belong to the interval leads to the inappropriate consequence; for example that at the begin or the end of the interval the state holds and holds not—is both, true and false. We have seen this difficulty before when considering open and closed sets (chapter 21). For events, it is customary to define them as semi-closed: the start point is part of the event, the endpoint is not, but is already the start point of the next event. This is also the solution the 'commonsense' world of everyday life has selected: a lesson from 4 to 5 starts at 4:00 and ends at 4:59 (chapter 8).

7. Special Case—Bounded, Linear Events

Some events are approximated with a linear function for a limited interval. This is appropriate for the interpolation of position of objects that move, or the outside temperature, etc. In this first step, we define the event as a single movement, from rest at *p* to rest at *q*, or a single raise of temperature; continuous movement from a through *b*, *c*, *d* to eventually *z* (Figure 604), or the rise of temperature during a day is a sequence of such bounded, linear events.

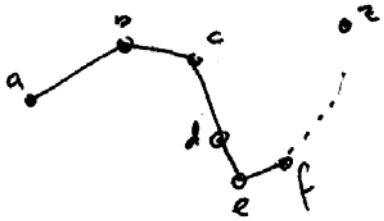


Figure 604: Movement of an object along a path

Outside of the interval, the value is not defined, which makes the function at a partial function. The interpolation for values inside the interval is a special case of linear interpolation and can be dealt with the methods described (see xx).

8. Movement of Extended Solid Objects

The movement of an extended object can be combined from a movement of the center of gravity and a rotation around this center (Figure 605). Rotations of objects are dealt with the same concept than translations of points: the angle of rotation is changing in time. The position and orientation of the object are two functions of time and the object geometry is transformed with the combined translation and rotation.

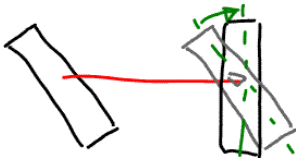


Figure 605: Object rotating and moving

$$G' = A \; G$$

$$G'(t) = A(t) \; G$$

$$A = \begin{vmatrix} \cos\alpha & -\sin\alpha & t_x \\ \sin\alpha & \cos\alpha & t_y \\ 0 & 0 & 1 \end{vmatrix}$$

$$A(t) = \begin{vmatrix} \cos\alpha(t) & -\sin\alpha(t) & t_x(t) \\ \sin\alpha(t) & \cos\alpha(t) & t_y(t) \\ 0 & 0 & 1 \end{vmatrix}$$

Using homogenous coordinates, the translation and the rotation can be expressed as matrices and the result of both is the product of the two matrices. For moving objects, these matrices are functions of time (compare with chapter 10):

9. Movement of Regions

Movement of regions does not imply that the form is maintained as it is the case for solid objects. A region can move and change form at the same time.

In this section we consider first the change in form caused by movements of the corners. We assume here that the region can be approximated before and after the move with a polygon with the same number of nodes.

The movement of the center of gravity is superimposed to a movement of the center of gravity and represented as changes relative to the center of gravity or which may be all what is represented of the movement of the region (Figure 606). The movement of corners relative to the center of gravity plus the movement of the center of gravity gives the total movement of each corner. This is an application of addition of movement, Figure 600).

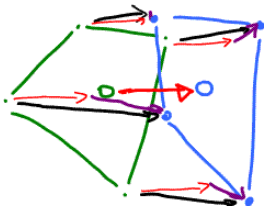


Figure 606: The movement of the center of gravity (red) plus the differential movement of the corners of the region (violet) give the total movement of each corner (black).

Assume that the region is represented by a polygon. Then the movement of the region is a movement of the corners. A region that is moving is thus nothing else than a polygon of moving points.

If the region changes form and therewith the number of corners changes, it is still a moving region, but not with a fixed set of corner points, but a changing set of corner points (Figure 607). The difference is only whether the functor is applied to the single points or to the polygon as a whole.

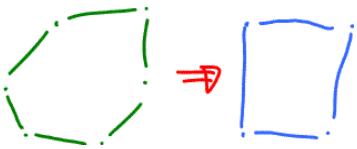


Figure 607: A polygon changing position and shape

10. Asynchronous Operations for Movements

Movements of point objects are important in life, but not always do we pay attention to all details. A number of abstractions from the complexity of the movement in time are used. Many of them can be seen as projects. For example, we identify the start and the end, the distance along the path or between start and end, the trajectory, which is independent of time. Intersections of trajectories are because they are potential points of interaction—desirable, when we meet in a restaurant to have lunch together, or undesirable, when cars collide. These operations with

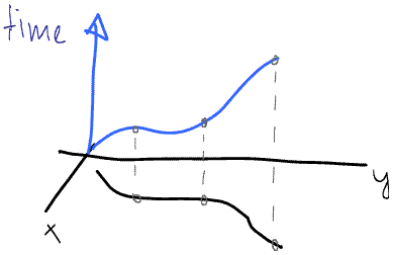


Figure 608: A point movement in x-y-time space and the corresponding trajectory in x-y

trajectories are asynchronous, they combine points the object passes through from different times.

10.1 Projections to space drop temporal aspects

The trajectory of a moving point is the path the point covers; it is the projection of the *2d* + time space to *2d* space, ignoring the temporal behavior. Take as an example the trajectory of airplanes (Figure 608).

The visualization as a space – time diagram (Figure 609) helps the intuition: in the *2d* plane we show location, in the third dimension, we show time. Moving points are then inclined lines, synchronous operations compare and combine points in the same (time) horizontal plane.

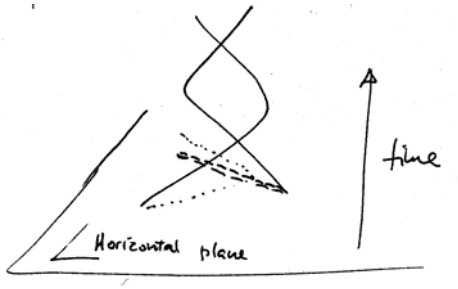


Figure 609 Space - time diagram

After projection into the space dimension we have the trajectories. We can see *intersection* of trajectories and we can ask questions like, how close did two trajectories ever come. Intersection of trajectories is not a collision, and the distance between two trajectories is not the same question as 'how close did two moving objects ever come'. The two trajectories in Figure 610 have an intersection point, but the two objects did pass at that point at different times, not colliding. The length of a trajectory is usually the length of the projection. In the projection we can also determine start and end points of a trajectory.

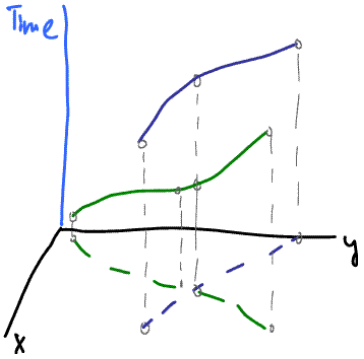


Figure 610: Two moving objects and their trajectories

Questions of whether a moving object did ever enter a region, or stayed completely within a region or was always outside of a region are also answered best when considering the trajectory (Figure 611). If the trajectory is closed, meaning the projection of start and end point are the same, then we can calculate the area enclosed.

Only a single operation to project a space-time path to the space is necessary. Then all the above described operations are operations with the resulting line, using previously defined geometric operations.

```
projectToSpace :: SpaceTimePath -> Line
```

Terminology
path a space time line
Trajectory the projection of a path

10.2 Other projections

A path has extreme points. For a flight path of an airplane, we can ask, where is the highest point and what height did the plane ever reach. Such questions are answered in other projections.

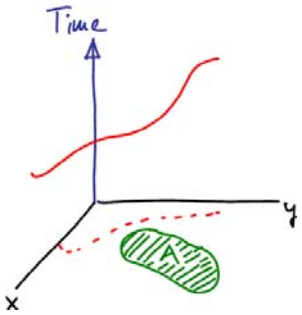


Figure 611: The animal did not enter the forest A

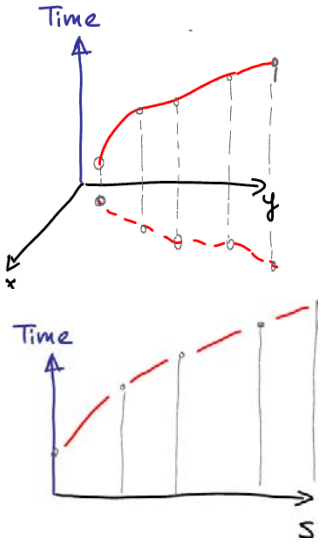


Figure 612: A path and a projection to the surface through it

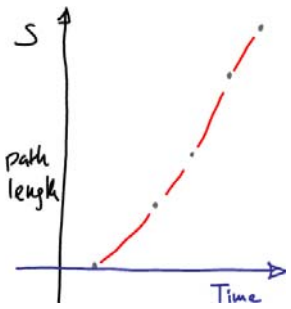


Figure 554-1: Perpendicular surface through path

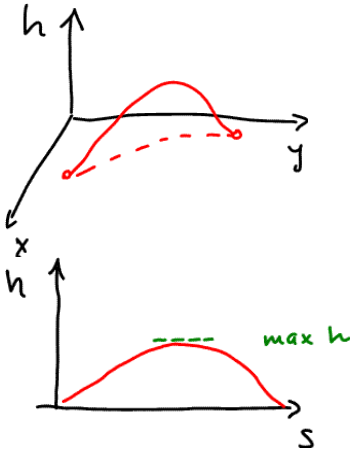


Figure 613: Highest point of a path

10.2.1 Projection to a surface along the trajectory

A path has a natural parameterization along the path by time or the length of the path. One can project the path to a surface perpendicular through the path (Figure 612). In this projection many question are immediately answerable:

Speed of movement is the derivative of the curve in the length-time space.

Highest and lowest points are maximum and minimum (Figure 613).

More complicated questions like: how long was the airplane between 1000 and 200 m above sea level can be answered. They are again as the intersection of this projection with regions.

11. SUMMARY

Storing data with type *changing object* in a database extends the database beyond the current snapshot database to a (world) temporal domain. We can not only ask questions like "where is object *O*" but also "Where was object *O* at time *T*" and a number of related questions. The functor *changing* is the methods to deal with moving objects and other changing aspects of an object. Projection from a path gives a trajectory. The structural strength of a functor allows us to use moving objects in complex queries wherever queries in a snapshot database refer to static objects.

12. REVIEW QUESTIONS

- Explain how the functor changing is applied to vector. What is the result?
- Why is changing vector of Float different from vector of changing float? Which one of the two is representing a movement?
- Demonstrate that the intersection point of two trajectories is not always the point where the distance between the two moving object is smallest.

Chapter 32 SPATIO-TEMPORAL DATABASES CONSTRUCTED WITH FUNCTORS

The extension of the current snapshot GIS to spatio-temporal data is a practical demand. The previous chapter has shown how to extend a database to cope with moving and changing objects.

In this chapter the requirements of administrative and legal procedures to establish when a fact was known is dealt with. In legal procedures it is not only when actions were performed and events occurred ("when did X kill Y, when did A sell property P to B"), but also, when did others obtain knowledge of a fact – when did I learn that A has sold his property. If the database shows only the state of the world at any given time, I cannot demonstrate later, what was known at the time I made a decision. Consider the decision of a bank employee to give a loan to *A* on March 1; he uses the database to check that *A* is the owner of property P that is used as collateral for the loan. Later, *A* default on the loan and the bank learns, that the property was sold to *B*, and this sale was completed on Feb. 25, which is prior to granting the loan and therefore the bank has no valid security for the loan. Has the employee made an error? No, on March 1, the database did show that *A* owns parcel *P*; the sale was recorded only on March 5 (and the mortgage on property *P* therefore in most countries valid). We see that it is not only to know when something happened, but also, when it was known, or became by registration 'public knowledge' (principle of giving notice).

A database must answer therefore two types of temporal questions:

- What was the case at time T (valid time)?
- What was known at time T (transaction time)?

In this short chapter, we show that this step is – with the preparation achieved now—simple: we apply the functor 'changing' twice to the database, once to obtain a database with values changing with the time in the world and once to obtain a database where previous states of the database can be retrieved to satisfy the 'giving notice' principle of administration.

1. INTRODUCTION

A temporal database must provide two time perspectives:

- Valid time, in which values describing reality are changing (sometimes called World time(Tansel, Clifford et al. 1993p.623), and
- Transaction time (sometimes called database time), in which the knowledge in the database is changing.

Temporal extension for data storage seem to face two major issues: a consistent and realistic calculus for intervals, including the special, ever changing constant 'now' (see xxx) and concept of a stable object.

A relational database can easily include a concept of user-time, which is a time, but without a defined semantics for the database; user-time is used to represent the time a snapshot was valid or to report time points like date of birth or date of hiring.

Note: temporal database literature uses the word event often as synonymous to instant or time point(Tansel, Clifford et al. 1993p.625), which is different from the definition of event as a interval for which a state obtains (see previous chapter).

The extension of a database which has objects with identity—relational or using another data model - to support time points and intervals of time is not difficult, it is mostly to construct a calculus for time points and time intervals.

The pure relational database cannot provide a stable object concept. The keys used to identify a tuple can change (Codd 1970) surrogates (Codd 1979) or time-invariant keys must be added to the model. For temporal relations additional, not well-understood rules of normalization seem necessary to avoid complications during updates. Much of the discussion argues for different types of granularity what changes with time: do we store changing relations (i.e., the full relation is time-stamped), changing tuples or changing values (Figure 614.), sometimes referred to as object versioning versus attribute versioning(Tansel, Clifford et al. 1993). This is primarily a question of implementation, which should not become visible at the user interface. Logically, time intervals for relations, tuples or single values are equivalent and can be transformed loss-less.

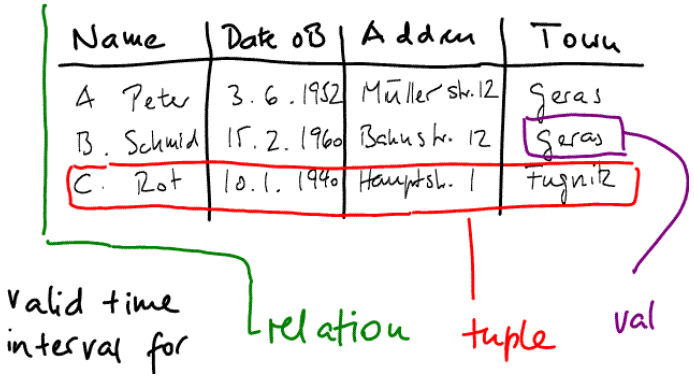


Figure 614: Different granularity for recording cahnge

2. CONCEPT OF TIME

A temporal database is built around a discrete time where a fixed granularity of time is assumed. "A *chronon* is the shortest duration of time supported"(Tansel, Clifford et al. 1993p.624). Such time points are isomorphic to integers and relate to Galton's discrete time (chapter 6). Wuu and Dayal (Tansel, Clifford et al. 1993) point out shortcomings and limitations caused by these assumptions and propose a concept of time that permits other specifications, for example non-metric or partial order.

We will use here time points that are isomorphic to integers and the previously defined algebra over intervals, which assume total order.

3. WORLD TIME PERSPECTIVE

The world is changing. We can differentiate two types of changes:

- new objects emerge and previously existing objects disappear, and
- property values of objects change.

The first type of change affects the lifespan of an object and the relevant changes are discussed under the heading lifestyle of objects, the second are changes in properties, including geometric properties of objects, and is dealt with using the functor changing (previous chapter).

Note: the term object in this section means the representation of something that is continuing in time. It is not necessarily a physical object.

3.1 LIFESPAN

Objects have a lifespan, a time in which they exist. The lifespan is an interval, in which the object representation is valid (Figure 615). After a record becomes invalid, it still exists in the

Figure 615: Lifespan of an object

database and its value can be accessed; care must be taken, that such 'old' object representations are not mixed with data that is maintained. Consider for example a database with employees for whom the address is stored: after an employee has quit, his last address is still available, but it is not likely updated. Here special support by the query languages is required to express a query to obtain the *last known fact*, separated from data that is *current*.

Technically, lifespans are asymmetric: before an object is created, nothing is known about it, not even that it will later exist. When the object is not-existing anymore, the data is still stored and it is possible to detect that the object has existed. This asymmetry is reflected in the implementation; if object identifiers are distributed in increasing order, then the test if an object O1 does exist consists of two tests

- Is O1 less than the highest assigned object identifier – if not, the object does not yet exist;
- Has O1 been destroyed, which is recorded in a relation? If not, then the object currently exists.

3.2 LIFESTYLES

The creation and destruction of objects is not the only two operations that can affect an object in its identity. Al-Taha and Barrera (Barrera, Frank et al. 1991) (Al-Taha and Barrera 1994) have identified a total of 11 situations that change the identity of an object.

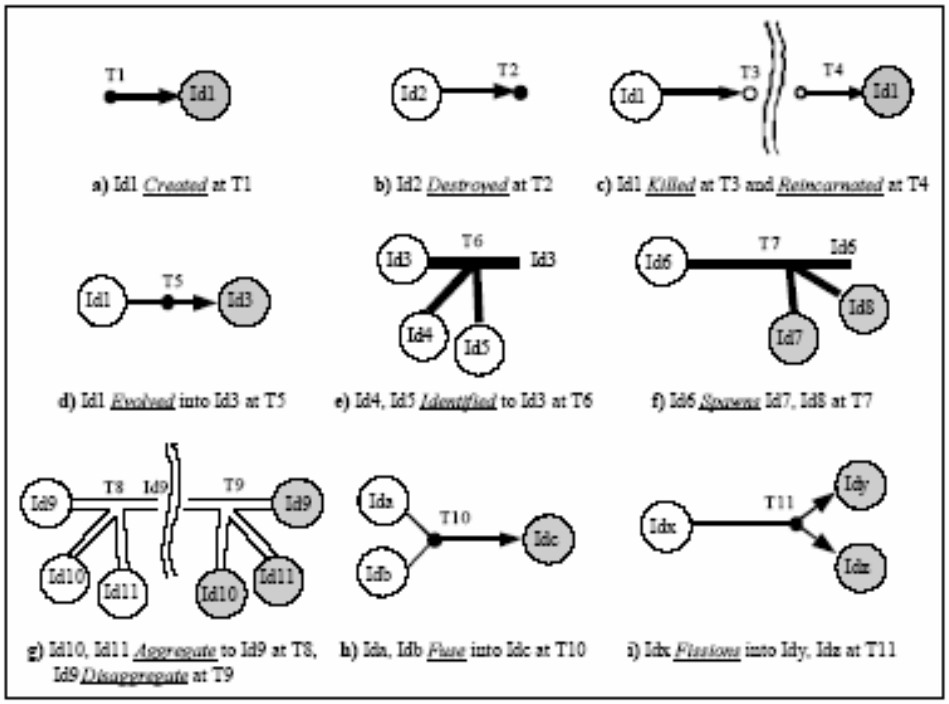


Figure 616: The 11 lifestyles(Al-Taha and Barrera 1994)

Not all these possible changes apply to all objects. Medak has identified lifestyles that restrict what changes are possible to certain ontological classes(Medak 1997; Medak 1999; Medak 2001). For example, liquids can be *identified* (poured together), but not *aggregated*, because liquids can be *spawned* (we can pour from a pitcher), but it is impossible to *disaggregate* two liquids once poured together. Similarly, for living things, it is ordinary not possible to *suspend* 'life' and *reincarnate* a person again (fairy tales and comic-books exempt), but a machine or car can be disassembled and does not exist as a whole and can be reassembled to exist again, which is either the changes *killed* and *reincarnated*, or *disaggregated* and *aggregated* (which is also not acceptable for living things—do you regularly disaggregate your cat?).

The lifestyle changes can be carried over from physical objects to geographic objects or objects created by social construction. Hornsby (Hornsby and Egenhofer 1997) has discussed lifestyles specifically for geographic objects like countries.

3.3 CHANGING VALUES

The properties describing the object, including the property 'existing', do change over the life of the object. These are changing values.

We have so far used the functor changing and applied it to values that changes continuously, but it is not restricted to this. Changing values can be of any type, including Boolean (Figure 617). A changing value of Boolean is true for some intervals and false for others (we have seen that it is an event—previous chapter); a changing Boolean can be converted in a sequence of intervals for which the value is true and an interval can be converted in a changing Boolean.

Figure 617: Changing Boolean

4. DATABASE OF CHANGING VALUES

Using parameterized types, the relation database used for snapshots of the world was a '*database of values*'. Applying the functor changing to values (as shown in the previous chapter), gives '*database of changing values*'.

The interpolation of administrative values is different from values for physical properties: physical properties change most often smoothly and we can interpolate between two states (Figure 618). Administrative facts are valid from a data till further notice (Figure 619).

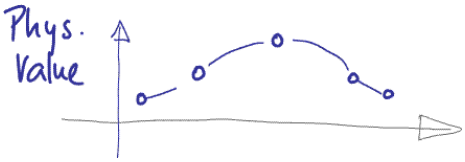


Figure 618:. Smoothly changing physical value

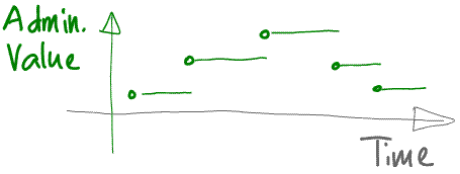


Figure 619: Stepwise change of administrative value

The query language (see chapter 16) remains the same, but returns now lists of changing values, from which the value for the time of interest is retrieved with the operation 'at' (see chapter 11). Selection of objects is now not with a single value, but a value and the time it is valid; instead of a condition to apply for the name of a town *("Geras" ==)*, we have to write *("Geras"==.at now*), where the condition 'name equals "Geras"' is composed with the conversion from a changing value to the value valid at time *now*.

5. DATABASE TIME

A database changes with time: new values are added, values are changed or objects are deleted. It is sometimes necessary to know in what state a database has been earlier—for example to determine if a user could have known a fact, or if he could have known if he had been careful. This is a principle of law and was introduced earlier as 'giving notice' (see example with bank granting a loan in the introduction).

The database itself is a changing value, which changes its value discretely and is valid from the change onwards till the next change. It behaves like administrative data (Figure 619) and the current state is valid, is the best knowledge till a new update

is received. This does not preclude that for certain smoothly changing value, an extrapolation in world time is possible (for example, airplanes moving), but this is the best 'current knowledge' and the extrapolation may change. At 11:00 we may ask where do we expect airplane at 12:00, if we receive an update in the airplane position at 11:15 and then ask at 11:20 again where we expect the airplane to be at 12:00, we get different information. In database that has a transaction time we can later ask "what was at 11:00 the expected location for the airplane at 12:00" and get the original estimate (Figure 620). Care must be used to separate extrapolated data from 'known' facts based on observations.

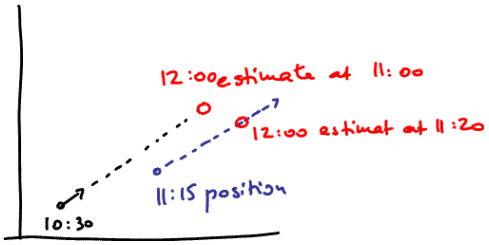


Figure 620: Observation of airplane at 10:30 and 11:15 with estimates for location at 12:00

The database is changing with every update. It is a changing value! The database perspective is achieved by applying the functor *changing* to the database as a whole: a temporal database with the database perspective is a *changing database of values*, a database with both time perspectives is a *changing database of changing values* (the functor *changing* applied twice).

Most queries will use the current state, but it must be possible to access previous states, the state as it was known at a previous time (our knowledge "as of March 1"). For these cases, the function '*as of time'* applied to the changing database returns a snapshot database for the indicated time, i.e., the database that was valid at that time. To this snapshot of the database, which is in the case of a bi-temporal database a database of changing values, i.e., a database with only world time perspective. To the result of the *'as of time' question,* a query with the *at* function can be applied. One can think of this two step execution of a query as two projections: first to the transaction time with *'as of* t_1*'* and then a second projection to valid time with *'at* t_2*'*.

6. Errors and Corrections

Databases can contain erroneous data. We have seen that internally with the use of logic, we can only ascertain that the database is consistent with respect to the rules fixed (see chapter 18). In a database with a database (transaction) time perspective, it could be possible to record if a change is inserting a new value or a change is the result of observing that a previously inserted value is in error(Tansel, Clifford et al. 1993). This will require two kinds of transactions, namely those that change values and those that correct values. I have not seen implementations of this

idea in database software, but the concept is included in instructions and laws, for example for the maintenance of land registries, where rules establish how new facts are recorded with reference to time and how special procedures are use to correct errors (again with a time annotation) (Schönenberger 1976)

PART TWELVE AFTERWARDS

This book has shown an eclectic collection of parts of mathematics. The inclusion of each piece was motivated by some function relevant for GIS applications. Many open problems of GIS research have been excluded, especially approximation, semantics and user interfaces, including graphical output. The material shown is the foundation onto which solutions to these difficult questions can be grafted.

1. FORMALITY LEADS TO CONSISTENCY

The approach I selected was very formal. My experience with the design of software is that the decisions early in the process have much effect later and errors in the beginning of a design are very difficult and very costly to correct later.

The same methods can serve many times in a GIS. In current commercial products, the same methods are implemented multiple times—justified (perhaps) with different optimizations—but with slightly different assumptions. These pieces are not consistent with each other. Extensive 'fixes' are later used to join the essentially similar but in detail dissimilar pieces. Coherences have been advocated, but difficult to achieve. It is necessary to clarify the foundation and to use systematic methods to combine modules.

The formal approach made the foundation evident and documented the decisions. Modules designed later are then linked to the previous decisions: the pieces are consistent and work together.

2. IS THAT ALL?

Are these all the parts? Is that all the mathematics necessary for a GIS? How could one demonstrate this? Can one proof the completeness?

I have advocated that "the proof is in the pudding" and started the implementation of the methods shown here. This can be useful to demonstrate that these methods are sufficient to solve a set of typical GIS application problems. This will

demonstrate that the core is covering what a GIS does (minus interface and nice output); it will also demonstrate, that the pieces are internally complete—what one module assumes another module provides.

An informal proof that the components shown are sufficient is using the rational decision mode (chapter 3): Every application of a GIS is to make a decision. A decision process can be formalized as a selection of the optimal variant. The following steps are necessary:

- Create all variants;
- Delete variants that are not acceptable based on properties of them:
- Evaluate each variant;
- Find variant with optimal value.

The methods to represent objects with spatio-temporal properties and to retrieve them have been shown in part 5. The construction of new geometries using vector operations was the topic of part 3 and 6. Properties of objects can be described with the methods in part 7 and 8, for networks in part 9. The evaluation combines methods from map algebra (part 4) and overlay computation (part 10). To sort the variants by evaluation value and pick the best is trivial.

3. CATEGORIES AND GIS THEORY

In a GIS many different parts of mathematics are combined. I have used here logic, algebra, set theory, topology, linear algebra to name but a few. Each comes with its own terminology and assumptions and the same theorems exist in different terminology in different part. The integration is difficult. I have used category theory as the unifying framework, in which all parts of mathematics of importance here can be integrated. This uses very little of category theory, but is sufficient to identify commonalities between some fields of mathematics and express them in a common language (see part 5).

All computer programs are functions that change the computer state (mostly the memory of the computer); they are in the category of Sets, where arrows are functions. This category is so dominant for implementation that I have assumed this category whenever no special category is used.

Different parts of mathematics use different categories, as is shown in the following table. Any implementation in a computer

program must translate a mathematical construction in some field and using some category to the category of sets and functions, because only these can be implemented. The language of category theory documents this transformation.

The commonality found in category theory is the key to identify the same constructions everywhere in a GIS and to establish consistency in the programs and unification of the seemingly disjoint mathematical theories applied.

Category	objects	Morphism	Part in this book
Set	sets	functions	Chapter 6 Measurements
Top	topological spaces	continuous functions	Part 7
Vect	vector spaces	linear transformations	Part 3: Space time, Part 6 Proj Space
Grp	groups	group homomorphism	
PO	partial ordered sets	monotone functions	
Graphs	edges and nodes	graph morphism	Part 8
Rel	relations	join	Part 5 DB

4. REVIEW

It came together nicely. The first parts were heavy, detailed, and sometimes painful. As compensation, the end was easy and swift. This justifies the hypothesis that a GIS is built from components. If the components are well-designed, they combine easily.

Let us review the components:

- The language and the conceptual framework: Algebra, second order functions, and category theory. This gave us functors that cover (nearly) all of spatial and temporal computations. We have found a generalization of map algebra to the temporal domain.
- Typed measurements and functions that connect them (like population density, connecting count of people with area). These functions were lifted to work with layers in a GIS, but also with time series—without additional new concepts for users of the GIS to learn.
- Simplification of data storage beyond the Relational Data Model, which itself is a considerable reduction in concepts and rules compared to the earlier Network Data Model. The

use of relations gives access to category theory, which results in a query language with only 2 essential elements, which are connected by function composition.

- An algebra of intervals and topological relations between them. This gives spatial and temporal topological predicates for a spatial query language but also the methods to express temporal conditions in the database query language.
- Projective geometry gives geometric computations without exceptions, which would then produce complication when combining with other concepts.
- Simplex and complex from combinatorial topology is the realm in which all geometric operations can be carried out. It gives a closed algebra for intersection and union of regions; it includes as a special case graphs and triangulations. Triangulations are the place where metric computations come together with combinatorial topology.
- Objects as the entities that continue in time and have changing attribute values (but not changing attributes).

I also think that I have achieved two steps forward for GIS:

- A bi-temporal GIS is constructed in a principled way by using the functor 'changing' for values, which gives the valid time perspective, and for the database as a whole, which gives the database time perspective.
- Unification of operations to apply for raster and vector representation alike; there are few areas where a full unification is not yet achieved and I give not up yet.

I conclude the first complete draft of this book on one of the last sunny fall days of the year: harvest time, leaves fall, apples are ripe and walnuts must be collected.

Geras, Oct. 17, 2004

BIBLIOGRAPHY

Abler, R. (1987). "The National Science Foundation - National Center for Geographic Information and Analysis." IJGIS **1**(4): 303-326.

Abler, R. (1987). Review of the Federal Research Agenda. International Geographic Information Systems (IGIS) Symposium (IGIS'87): The Research Agenda, Arlington, VA.

Abler, R., J. S. Adams, et al. (1971). Spatial Organization - The Geographer's View of the World. Englewood Cliffs, N.J., USA, Prentice Hall.

ADA (1983). ADA Reference Manual for the ADA Programming Language. New York, NY, Springer-Verlag.

Adam, J. (1982). "A Detailed Study of the Duality Relation for the Least Squares Adjustment in Euclidean Spaces." Bulletin Géodésique **56**: 180 - 195.

Adams, D. (2002). The Ultimate Hitchhiker's Guide to the Galaxy, Del Rey.

Adams, J. L. (1979). Conceptual Blockbusting. New York, Norton & Company.

Al-Taha, K. (1992). Temporal Reasoning in Cadastral Systems, University of Maine.

Al-Taha, K. and R. Barrera (1994). Identities through Time. International Workshop on Requirements for Integrated Geographic Information Systems, New Orleans, Louisiana.

Alexander, C., S. Ishikawa, et al. (1977). A Pattern Language - Towns, Buildings, Construction. New York, Oxford University Press.

Alexandroff, P. (1961). Elementary Concepts of Topology. New York, USA, Dover Publications.

Allen, J. and P. Hayes (1985). "A Common-Sense Theory of Time." IJCAI: 528 - 531.

Allen, J. F. (1981). A General Model of Action and Time. New York, NY, Department of Computer Science, University of Rochester.

Allen, J. F. (1983). "Maintaining Knowledge about Temporal Intervals." Communications of the ACM **26**(11): 832-843.

Allen, J. F. (1984). "Towards a General Theory of Action and Time." Artificial Intelligence **23**.

Allen, J. F. and H. A. Kautz (1985). A Model of Naive Temporal Reasoning. J.R.Hobbs and R.C.Moore (Eds.)://Formal Theories of the Commonsense World//Ablex Publishing Company, orig.

ANSI X3/SPARC (1975). "Study Group on Database Management Systems, Interim Report 75-02-08." SIGMOD **7**(2).

ANSI X3H2 (1985). American National Standard Database Language SQL, American National Standards Database Committee.

Asimov, I. (1957). Earth is Room Enough. New York, Doubleday.

Asperti, A. and G. Longo (1991). Categories, Types and Structures - An Introduction to Category Theory for the Working Computer Scientist. Cambridge, Mass., The MIT Press.

Atkinson, M., F. Bancilhon, et al. (1989). The Object-Oriented Database System Manifesto. First International Conference on Deductive and Object-Oriented Databases, Elsevier.

Bachman, C. W. (1973). "The Programmer as Navigator." Communic of ACM Communications of the ACM **16**.

Backus, J. (1978). "Can Programming Be Liberated from the von Neumann Style? A Functional Style and Its Algebra of Programs." CACM **21**: 613-641.

Baird, D. G., R. H. Gertner, et al. (1994). Game Theory and the Law. Cambridge, Mass., Harvard University Press.

Bancilhon, F., C. Delobel, et al. (1992). Building an Object-Oriented Database System - The Story of O_2. San Mateo, CA, Morgan Kaufmann.

Barr, M. and C. Wells (1990). Category Theory for Computing Science. New York, Prentice Hall.

Barrera, R., A. U. Frank, et al. (1991). "Temporal Relations in Geographic Information Systems: A Workshop at the University of Maine." SIGMOD Record **20**(3): 85-91.

Batty, M. and P. Longley (1994). Fractal Cites: A Geometry of Form and Function. London, Academic Press Limited.

Beard, K. (1988). Multiple Representations from a Detailed Database: a Scheme for Automated Generalization, University of Wisconsin - Madison.

Bertalanffy, L. v. (1973). General System Theory: Foundations, Development, Applications (Penguin University Books), Penguin Books.

Bertin, J. (1977). La Graphique et le Traitement Graphique de l'Information. Paris, Flammarion.

Bird, R. (1998). Introduction to Functional Programming Using Haskell. Hemel Hempstead, UK, Prentice Hall Europe.

Bird, R. and O. de Moor (1997). Algebra of Programming. London, Prentice Hall Europe.

Bird, R. and P. Wadler (1988). Introduction to Functional Programming. Hemel Hempstead, UK, Prentice Hall International.

Birkhoff, G. (1967). Lattice Theory. Providence, RI, American Mathematical Society (Colloquium Publications).

Bittner, T. (1999). Rough Location. Institute of Geoinformation. Vienna, Austria, Technical University**:** 196.

Bittner, T. and A. U. Frank (1997). An Introduction to the Application of Formal Theories to GIS. Angewandte Geographische Informationsverarbeitung IX (AGIT), Salzburg, Institut fuer Geographie, Universitaet Salzburg.

Bittner, T. and B. Smith (2003). Directly Depicting Granular Ontologies. IFOMIS, NCGIA. Leipzig, Buffalo, Univerity of Leipzig, University at Buffalo**:** 15.

Bittner, T. and B. Smith (2003). Granular Spatio-Temporal Ontologies. 2003 AAAI Symposium: Foundations and Applications of Spatio-Temporal Reasoning (FASTR)**:** 6.

Bittner, T. and B. Smith (2003 (draft)). Formal Ontologies for Space and Time. IFOMIS, Department of Philosophy. Leipzig, Buffalo, University of Leipzig, University at Buffalo and NCGIA**:** 17.

Björner, A. (1999). Oriented Matroids. Cambridge, England, Cambridge University Press.

Blumenthal, L. M. (1986). A Modern View of Geometry. New York, Dover Publications, Inc.

Blumenthal, L. M. and K. Menger (1970). Studies in Geometry. San Francisco, W. H. Freeman and Company.

Booch, G., J. Rumbaugh, et al. (1997). Unified Modeling Language Semantics and Notation Guide 1.0. San Jose, CA, Rational Software Corporation.

Borges, P. R. (1997). Sequence Implementations in Haskell. Computing Laboratory. Oxford, Oxford University.

Brodie, M. L., J. Mylopoulos, et al. (1984). On Conceptual Modelling: Perspectives from Artificial Intelligence, Databases, and Programming Languages, Springer-Verlag.

Buchmann, A., O. Günther, et al., Eds. (1990). Design and Implementation of Large Spatial Databases - Proceedings of the First Symposium SSD '89, Santa Barbara, CA, July 1989. Lecture Notes in Computer Science. Berlin, Springer-Verlag.

Bugayevskiy, L. M. and J. P. Snyder (1995). Map Projections - A Reference Manual. London, Taylor & Francis.

Burrough, P. A. and A. U. Frank (1995). "Concepts and Paradigms in Spatial Information: Are Current Geographic Information Systems Truly Generic?" International Journal of Geographical Information Systems **9**(2): 101-116.

Burrough, P. A. and A. U. Frank, Eds. (1996). Geographic Objects with Indeterminate Boundaries. GISDATA Series. London, Taylor & Francis.

Buttenfield, B. P. (1984). Line Structures in Graphic and Geographic Space, University of Washington.

Buttenfield, B. P. (1989). "Scale-Dependence and Self-Similarity of Cartographic Lines." Cartographica **26**(1): 79-100.

Buttenfield, B. P. (1993). Multiple Representations - Closing Report. Buffalo, State University of New York at Buffalo.

Buttenfield, B. P. and J. Delotto (1989). Multiple Representations: Report on the Specialist Meeting, National Center for Geographic Information and Analysis; Santa Barbara, CA.

Cardelli, L. (1997). Type Systems. Handbook of Computer Science and Engineering. A. B. Tucker, CRC Press**:** 2208-2236.

Carnap, R. (1958). Introduction to Symbolic Logic and its Applications. New York, Dover Publications.

Caroll, L. (1893). Sylvie and Bruno. London, Macmillan.

Casati, R. and A. C. Varzi (1994). Holes and Other Superficialities. Cambridge, Mass., MIT Press.

Caschetta, A. R. (2000). First International Conference on Geographic Information Science. Savannah, Georgia, USA, University of California Regents.

Chang, S.-K., E. Jungert, et al. (1990). The Design of Pictorial Databases Based Upon the Theory of Symbolic Projection. Design and Implementation of Large Spatial Databases. A. Buchmann, O. Günther, T. R. Smith and Y.-F. Wang. New York, NY, Springer-Verlag. **409:** 303-324.

Chang, S. K., E. Jungert, et al. (1989). Representation and Retrieval of Symbolic Pictures Using Generalized 2D Strings. SPIE Visual Communications and Image Processing Conference.

Chen, P. P.-S. (1976). "The Entity-Relationship Model - Toward a Unified View of Data." ACM Transactions on Database Systems **1**(1): 9 - 36.

Chomsky, N. (1980). Rules and Representations. The Behavioral and Brain Sciences. **3:** 1 - 61.

Chrisman, N. (1997). Exploring Geographic Information Systems. New York, John Wiley.

Chrisman, N., J. A. Dougenik, et al. (1992). Lessons for the Design of Polygon Overlay Processing from the ODYSSEY WHIRLPOOL Algorithm. Proceedings of the 5th International Symposium on Spatial Data Handling, Charleston, IGU Commission of GIS.

Chrisman, N. R. (1975). Topological Information Systems for Geographic Representation. Proc. Auto Carto 2//Reston, VA, 1975, x.

Christaller, W. (1966). Central Places in Southern Germany. Englewood Cliffs, NJ, Prentice Hall.

Clementini, E. and P. Di Felice (1996). An Algebraic Model for Spatial Objects with Indeterminate Boundaries. Geographic Objects with Indeterminate Boundaries European Science Foundation. P. A. Burrough and A. U. Frank, Taylor & Francis. **2:** 155-169.

Clocksin, W. F. and C. S. Mellish (1981). Programming in Prolog. Berlin, Springer-Verlag.

CODASYL (1971). Data Base Task Group Report.

CODASYL (1971). Report of the Data Base Task Group.

Codd, E. (1979). "Extending the Database Relational Model to Capture More Meaning." ACM TODS **4**(4): 379-434.

Codd, E. F. (1970). "A Relational Model of Data for Large Shared Data Banks." Communications of the ACM **13**(6): 377 - 387.

Codd, E. F. (1982). "Relational Data Base: A Practical Foundation for Productivity." Communications of the ACM **25**(2): 109-117.

Codd, E. F. (1991). The Relational Model for Database Management. Reading, Mass., Addison-Wesley.

Cohn, A. G. (1995). A Hierarchical Representation of Qualitative Shape Based on Connection and Convexity. Spatial Information Theory - A Theoretical Basis for GIS (Int. Conference COSIT'95). A. U. Frank and W. Kuhn. Berlin, Springer-Verlag. **988:** 311-326.

Cohn, A. G. and N. M. Gotts (1996). The 'Egg-Yolk' Representation of Regions with Indeterminate Boundaries. Geographic Objects with Indeterminate Boundaries. P. Burrough and A. U. Frank. London, Taylor & Francis. **GISDATA II**.

Cohn, A. G. and S. M. Hazarika (2001). "Qualitative Spatial Representation and Reasoning: An Overview." Fundamenta Informaticae: 2-32.

Colmerauer, A., H. Kanoui, et al. (1983). Prolog, bases théoriques et développements actuels. Technique et Science Informatiques//vol. 2, No. 4//pp. 271 - 311, copy with Frank.

Corbett, J. (1975). Topological Principles in Cartography. 2nd International Symposium on Computer-Assisted Cartography, Reston, VA.

Couclelis, H. (1992). People Manipulate Objects (but Cultivate Fields): Beyond the Raster-Vector Debate in GIS. Theories and Methods of Spatio-Temporal Reasoning in Geographic Space. A. U. Frank, I. Campari and U. Formentini. Berlin, Springer-Verlag. **639:** 65-77.

Couclelis, H. and N. Gale (1986). "Space and Spaces." Geografiska Annaler **68**(1): 1-12.

Date, C. J. (1983). An Introduction to Database Systems. Reading, MA, Addison-Wesley.

Davis, M. D. (1983). Game Theory. Minneola, NY, Dover Publications.

DEC (1974). COGO-10 Reference Manual. Digital Equipment Corporation, KEDV.

Deux, O. (1989). The Story of O2. Fifth Conference on Data and Knowledge Engineering.

Dijkstra, E. W. (1959). "A Note on Two Problems in Connection with Graphs." Numerische Mathematik(1): 269-271.

Dutton, G., Ed. (1978). First International Advanced Study Symposium on Topological Data Structures for Geographic Information Systems. Harvard Papers on Geographic Information Systems. Reading, Mass., Addison-Wesley.

Dutton, G., Ed. (1979). First International Study Symposium on Topological Data Structures for Geographic Information Systems (1977). Harvard Papers on Geographic Information Systems. Cambridge, MA, Harvard University.

Eastman, J. R. (1993). IDRISI Technical Reference. Worcester, Mass., Clark University.

Edelsbrunner, H. (2001). Geometry and Topology for Mesh Generation. Cambridge, England, Cabridge University Press.

Egenhofer, M. (1993). "What's Special about Spatial - Database Requirements for Vehicle Navigation in Geographic Space." SIGMOD Record **22**(2): 398-402.

Egenhofer, M. and A. U. Frank (1992). "Object Oriented Modeling for GIS." Journal of the Urban and Regional Information Systems URISA **4**(2): 3-19.

Egenhofer, M. J. (1989). Spatial Query Languages, University of Maine.

Egenhofer, M. J. and K. K. Al-Taha (1992). Reasoning About Gradual Changes of Topological Relationships. Theories and Methods of Spatio-Temporal Reasoning in Geographic Space. A. U. Frank, I. Campari and U. Formentini. Heidelberg-Berlin, Springer-Verlag. **639:** 196-219.

Egenhofer, M. J., E. Clementini, et al. (1994). "Topological Relations between Regions with Holes." IJGIS **8**(2): 129-142.

Egenhofer, M. J. and A. U. Frank (1992). User Interfaces for Spatial Information Systems: Manipulating the Graphical Representation. Geologisches Jahrbuch. R. Vinken. **A 122:** 59-69.

Egenhofer, M. J. and R. D. Franzosa (1995). "On the Equivalence of Topological Relations." International Journal for Geographical Information Systems **9**(2): 133-152.

Egenhofer, M. J. and R. G. Golledge (1994). Time in Geographic Space: Report on the Specialist Meeting of Research Initiative 10. Santa Barbara, CA, National Center for Geographic Information and Analysis.

Egenhofer, M. J. and J. R. Herring (1991). Categorizing Binary Topological Relationships Between Regions, Lines, and Points in Geographic Databases, Department of Surveying Engineering, University of Maine, Orono, ME.

Egenhofer, M. J. and D. M. Mark (1995). "Modelling Conceptual Neighbourhoods of Topological Line-Region Relations." International Journal for Geographical Information Systems **9**(5): 555-565.

Egenhofer, M. J. and J. Sharma (1992). Topological Consistency. Proceedings of the 5th International Symposium on Spatial Data Handling, Charleston, IGU Commission of GIS.

Egenhofer, M. J., J. Sharma, et al. (1993). A Critical Comparison of the 4-Intersection and 9-Intersection Models for Spatial Relations: Formal Analysis. Autocarto 11, Minneapolis, ACM/ASPRS.

Ehrich, H.-D. (1981). Specifying Algebraic Data Types by Domain Equations. Forschungsbericht Nr. 109//Universität Dortmund//Abteilung Informatik, P714898:109.

Ehrich, H.-D., M. Gogolla, et al. (1989). Algebraische Spezifikation abstrakter Datentypen. Stuttgart, B.G. Teubner.

Ehrig, H. and B. Mahr (1985). Fundamentals of Algebraic Specification. Berlin, Springer-Verlag.

Eichhorn, G., Ed. (1979). Landinformationssysteme. Schriftenreihe Wissenschaft und Technik. Darmstadt, Germany, TH Darmstadt.

ESRI (1993). Understanding GIS - The ARC/INFO Method. Harlow, Longman; The Bath Press.

Everling, W. (1987). "Temporal Logic." Informatik-Spektrum **10**(2): 99-100.

Fagan, G. and H. Soehngen (1987). Improvement of GBF/DIME file coordinates in a geobased information system by various transformation methods and "rubbersheeting" based on triangulation. Auto-Carto 8, Baltimore, MA, ASPRS & ACSM.

Faugeras, O. (1993). Three-Dimensional Computer Vision, The MIT Press.

Faugeras, O. and Q.-T. Luong (2001). The Geometry of Multiple Images. Cambridge, Massachusetts and London, England, The MIT Press.

Fellbaum, C., Ed. (1998). WordNet: An Electronic Lexical Database. Language, Speech, and Communication. Cambridge, Mass., The MIT Press.

Flewelling, D. M., M. J. Egenhofer, et al. (1992). Constructing Geological Cross Sections with a Chronology of Geologic Events. 5th International Symposium on Spatal Data Handling, Charleston, South Carolina, USA, IGU Commision on GIS.

Fonseca, F. T., M. J. Egenhofer, et al. (2002). "Using Ontologies for Integrated Geographic Information Systems." Transactions in GIS **6**(3): 231-57.

Förstner, W. and B. Wrobel (Draft). Digitale Photogrammetrie, Springer.

Foyley, J. D. and A. van Dam (1982). Fundamentals of Interactive Computer Graphics. Reading MA, Addison-Wesley Publ. Co.

Franck, G. (1998). Ökonomie der Aufmerksamkeit. München Wien, Carl Hanser Verlag.

Frank, A. (1983). Datenstrukturen für Landinformationssysteme - Semantische, Topologische und Räumliche Beziehungen in Daten der Geo-Wissenschaften. Institut für Geodäsie und Photogrammetrie, ETH Zürich.

Frank, A. and E. Grum, Eds. (2004). Proceedings of the ISSDQ '04 Vol 1. Geoinfo Series. Vienna, Austria, Institute for Geoinformation.

Frank, A. and E. Grum, Eds. (2004). Proceedings of the ISSDQ '04 Vol 2. Geoinfo Series. Vienna, Austria, Institute for Geoinformation.

Frank, A. U. (1981). Application of DBMS to Land Information Systems. Seventh International Conference on Very Large Data Bases VLDB, Cannes, France.

Frank, A. U. (1982). "MAPQUERY: Database Query Language for Retrieval of Geometric Data and its Graphical Representation." ACM SIGGRAPH **16**(3): 199 - 207.

Frank, A. U. (1984). "Computer Assisted Cartography - Are We Treating Graphics or Geometry?" Journal of Surveying Engineering **110**(2): 159-168.

Frank, A. U. (1985). Computer Assisted Cartography, Lecture Notes, Surveying Engineering Program, University of Maine at Orono.

Frank, A. U. (1985). "Computer Education for Surveyors." Canadian Surveyor **39**(4): 323-331.

Frank, A. U. (1985). Course Notes for SVE 451, Geographic Information Systems. Orono, ME, University of Maine.

Frank, A. U. (1988). Requirements for a Database Management System for a GIS. PE & RS.

Frank, A. U. (1988). Requirements for Database Management Systems Used for AM/FM Data. AM/FM Snowmass Conference, Snowmass, CO, AM/FM International.

Frank, A. U. (1990). Spatial Concepts, Geometric Data Models and Data Structures. GIS Design Models and Functionality, Leicester, UK, Midlands Regional Research Laboratory, University of Leicester.

Frank, A. U. (1991). Properties of Geographic Data: Requirements for Spatial Access Methods. Advances in Spatial Databases - 2nd Symposium on Large Spatial Databases, SSD'91 (Zurich, Switzerland). O. Guenther and H.-J. Schek. Berlin-Heidelberg, Springer-Verlag. **525:** 225-233.

Frank, A. U. (1994). Qualitative Temporal Reasoning in GIS - Ordered Time Scales. Sixth International Symposium on Spatial Data Handling, SDH'94, Edinburgh, Scotland, Sept. 5-9, 1994, IGU Commission on GIS.

Frank, A. U. (1995). History. Geographic Information Systems - Materials for a Post-Graduate Course; Vol. 3: GIS Organization GeoInfo Series. A. U. Frank. Vienna, Austria, Dept. of Geoinformation, TU Vienna. **6**.

Frank, A. U. (1996). The Prevalence of Objects with Sharp Boundaries in GIS. Geographic Objects with Indeterminate Boundaries. P. A. Burrough and A. U. Frank. London, Taylor & Francis. **II:** 29-40.

Frank, A. U. (1998). Different Types of 'Times' in GIS. Spatial and Temporal Reasoning in GIS. M. J. Egenhofer and R. G. Golledge. New York, Oxford University Press**:** 40-61.

Frank, A. U. (1998). GIS for Politics. GIS Planet '98, Lisbon, Portugal (9 - 11 Sept. 1998), IMERSIV.

Frank, A. U. (1999). One Step up the Abstraction Ladder: Combining Algebras - From Functional Pieces to a Whole. Spatial Information Theory - Cognitive and Computational Foundations of Geographic Information Science (Int. Conference COSIT'99, Stade, Germany). C. Freksa and D. M. Mark. Berlin, Springer-Verlag. **1661:** 95-107.

Frank, A. U. (2001). "Tiers of Ontology and Consistency Constraints in Geographic Information Systems." International Journal of Geographical Information Science **75**(5 (Special Issue on Ontology of Geographic Information)): 667-678.

Frank, A. U. (2003). Ontology for Spatio-Temporal Databases. Spatiotemporal Databases: The Chorochronos Approach. M. Koubarakis, T. Sellis and e. al. Berlin, Springer-Verlag**:** 9-78.

Frank, A. U. (to appear). Ontology for Geoinformation.

Frank, A. U. and I. Campari, Eds. (1993). Spatial Information Theory - Theoretical Basis for GIS (European Conference on Spatial Information Theory COSIT'93). Lecture Notes in Computer Science. Berlin-Heidelberg, Springer-Verlag.

Frank, A. U., I. Campari, et al., Eds. (1992). Theories and Methods of Spatio-Temporal Reasoning in Geographic Space. Lecture Notes in Computer Science 639. Pisa, Italy, Springer Verlag.

Frank, A. U., D. L. Hudson, et al. (1987). Artificial Intelligence Tools for GIS. International Geographic Information Systems (IGIS) Symposium: The Research Agenda, Crystal City, VA, NASA.

Frank, A. U. and W. Kuhn (1986). Cell Graph: A Provable Correct Method for the Storage of Geometry. Second International Symposium on Spatial Data Handling, Seattle, WA.

Frank, A. U. and D. M. Mark (1991). Language Issues for Geographical Information Systems. Geographic Information Systems: Principles and Applications. D. Maguire, D. Rhind and M. Goodchild. London, Longman Co.

Frank, A. U., B. Palmer, et al. (1986). Formal Methods for Accurate Definition of Some Fundamental Terms in Physical Geography. Second International Symposium on Spatial Data Handling, Seattle, Wash.

Frank, A. U., J. Raper, et al., Eds. (2001). Life and Motion of Socio-Economic Units. GISDATA Series. London, Taylor & Francis.

Frank, A. U. and M. Raubal (1998). Specifications for Interoperability: Formalizing Image Schemata for Geographic Space. SDH'98, Vancouver, Canada.

Frank, A. U. and M. Raubal (1999). "Formal Specifications of Image Schemata - A Step to Interoperability in Geographic Information Systems." Spatial Cognition and Computation **1**(1): 67-101.

Frank, A. U. and V. Robinson (1987). "Expert Systems for Geographic Information Systems." Photogrammetric Engineering and Remote Sensing**?52**(10, Oct.).

Frank, A. U., V. Robinson, et al. (1986). "An Assessment of Expert Systems Applied to Problems in Geographic Information Systems." ASCE Journal of Surveying Engineering **112**(3).

Frank, A. U., V. Robinson, et al. (1986). "Expert Systems for Geographic Information Systems: Review and Prospects." Surveying and Mapping **112**(2): 119-130.

Frank, A. U., V. Robinson, et al. (1986). "An Introduction to Expert Systems." ASCE Journal of Surveying Engineering **112**(3).

Frank, A. U. and B. Studenmann (1983). Semantische, topologische und räumliche Datenstrukturen in Landinformationssystemen. FIG XVII. Congress, Sofia, Bulgarien 1983.

Frank, A. U. and S. Timpf (1994). "Multiple Representations for Cartographic Objects in a Multi-Scale Tree - An Intelligent Graphical Zoom." Computers and Graphics Special Issue on Modelling and Visualization of Spatial Data in GIS **18**(6): 823-829.

Frank, A. U., G. S. Volta, et al. (1997). "Formalization of Families of Categorical Coverages." IJGIS **11**(3): 215-231.

Freksa, C. and D. M. Mark, Eds. (1999). Spatial Information Theory (Int. Conference COSIT'99, Stade, Germany). Lecture Notes in Computer Science. Berlin, Springer-Verlag.

Gallaire, H. (1981). Impacts of Logic on Data Bases. Proc. 7th International Conf. on VLDB, Cannes.

Gallaire, H., J. Minker, et al. (1984). "Logic and Databases: A Deductive Approach." ACM **16**(2): 153-184.

Galton, A., Ed. (1987). Temporal Logics and Their Applications, Academic Press.

Galton, A. (1997). Continuous Change in Spatial Regions. Spatial Information Theory - A Theoretical Basis for GIS (International Conference COSIT'97). S. C. Hirtle and A. U. Frank. Berlin-Heidelberg, Springer-Verlag. **1329:** 1-14.

Galton, A. (2000). Qualitative Spatial Change. Oxford, Oxford University Press.

Gamma, E., R. Helm, et al. (1995). Design Patterns, Addison-Wesley Professional.

Gill, A. (1976). Applied Algebra for the Computer Sciences. Englewood Cliffs, NJ, Prentice-Hall.

Goguen, J. A., J. W. Thatcher, et al. (1975). Abstract Data Types as Initial Algebras and Correctness of Data Representations. Conf. on Computer Graphics, Pattern Recognition and Data Structures, May 1975.

Goldenhuber, C. (1997). Aufdeckung von Numerischen Problemen in geodätischer Software. Vienna, Austria, Institute for Geoinformation.

Goodchild, M. and R. Jeansoulin, Eds. (1998). Data Quality in Geographic Information - From Error to Uncertainty. Paris, Hermes.

Goodchild, M. F. (1990). A Geographical Perspective on Spatial Data Models. GIS Design Models and Functionality, Leicester, Midlands Regional Research Laboratory.

Goodchild, M. F. (1990). Spatial Information Science. 4th International Symposium on Spatial Data Handling, Zurich, Switzerland (July 23-27, 1990), International Geographical Union, Commission on Geographic Information Systems.

Goodchild, M. F. (1992). "Geographical Data Modeling." Computers and Geosciences **18**(4): 401- 408.

Goodchild, M. F. (1992). "Geographical Information Science." International Journal of Geographical Information Systems **6**(1): 31-45.

Goodchild, M. F., M. J. Egenhofer, et al. (1999). "Introduction to the Varenius Project." International Journal of Geographical Information Science **13**(8): 731-745.

Goodchild, M. F. and S. Gopal (1990). The Accuracy of Spatial Databases. London, Taylor & Francis.

Gray, J. and A. Reuter (1993). Transaction Processing: Concepts and Techniques. San Francisco, CA, Morgan Kaufmann.

Guibas, L. J. and J. Stolfi (1982). "A Language for Bitmap Manipulation." ACM Transactions on Graphics **1**(3).

Guibas, L. J. and J. Stolfi (1987). Ruler, Compass and Computer//The Design and Analysis of Geometric Algorithms. Theoretical Foundations of Computer Graphics and CAD, Il Ciocco, Italy, NATO Advanced Study Institute.

Günther, O. (1989). Database Support for Multiple Representation. Multiple Representations: Initiative 3 Specialist Meeting Report. B. P. Buttenfield and J. S. DeLotto. Santa Barbara, CA, NCGIA. **89-3:** 50-52.

Guttag, J. V. and J. J. Horning (1978). "The Algebraic Specification of Abstract Data Types." Acta Informatica **10**(1): 27-52.

Guttag, J. V., J. J. Horning, et al. (1985). Larch in Five Easy Pieces, Digital Equipment Corporation, Systems Research Center.

Haerder, T. and A. Reuter (1983). "Principles of Transaction-Oriented Database Recovery." ACM Computing Surveys **15**(4 (December 1983)).

Hartley, R. and A. Zisserman (2000). Multiple View Geometry in Computer Vision, Cambridge University Press.

Hartley, R. and A. Zisserman (2003). Multiple View Geometry in Computer Vision. Cambridge, UK, Cambridge University Press.

Heath, T. L. (1981). History of Greek Mathematics: From Thales to Euclid, Dover Publications.

Henle, M. (1994). A Combinatorial Introduction to Topology. New York, Dover Publications.

Herring, J., M. J. Egenhofer, et al. (1990). Using Category Theory to Model GIS Applications. 4th International Symposium on Spatial Data Handling, Zurich, Switzerland, IGU, Commission on Geographic Information Systems.

Herring, J. R. (1987). TIGRIS: Topologically Integrated Geographic Information System. Auto-Carto 8, Baltimore, MA, ASPRS & ACSM.

Herring, J. R. (1990). TIGRIS: A Data Model for an Object Oriented Geographic Information System. GIS Design Models and Functionality, Leicester, Midlands Regional Research Laboratory.

Herring, J. R. (1991). The Mathematical Modeling of Spatial and Non-Spatial Information in Geographic Information Systems. Cognitive and Linguistic Aspects of Geographic Space: An Introduction. D. M. Mark and A. U. Frank. Dordrecht, Kluwer Academic**:** 313-350.

Heuvelink, G. B. M. (1998). Error Propagation in Environmental Modelling with GIS. London, Taylor & Francis.

Hillier, B. (1999). Space Is the Machine, Cambridge University Press.

Hillier, B. and J. Hanson (1984). The Social Logic of Space. Cambridge, Cambridge University Press.

Hofstadter, D. R. (1985). Gödel, Escher, Bach - ein Endloses Geflochtenes Band. Stuttgart, Ernst Klett Verlag.

Horn, B. K. P. (1986). Robot Vision. Cambridge, Mass, MIT Press.

Hornsby, K. and M. J. Egenhofer (1997). Qualitative Representation of Change. Spatial Information Theory - A Theoretical Basis for GIS (International Conference COSIT'97). S. C. Hirtle and A. U. Frank. Berlin-Heidelberg, Springer-Verlag. **1329:** 15-33.

Hrbek (1993). 70 Jahre Bundesamt für Eich- und Vermessungswesen. Wien, Manz.

Hudak, P., J. Peterson, et al. (1997). A Gentle Introduction to Haskell, Yale University.

ISO. (2004). "ISO/TC 211 Geographic information/Geomatics." Retrieved 11. 09., 2004, from http://www.isotc211.org/.

Jensen, K. and N. Wirth (1975). PASCAL User Manual and Report. Berlin-Heidelberg, Springer-Verlag.

Jones, C. B., G. L. Bundy, et al. (1995). "Map generalization with a triangulated data structure." CaGIS **22**(4): 317-331.

Jungert, E. (1992). The Observer's Point of View: An Extension of Symbolic Projection. Theories and Methods of Spatio-Temporal Reasoning in Geographic Space. A. U. Frank, I. Campari and U. Formentini. Heidelberg-Berlin, Springer-Verlag. **639:** 179-195.

Jungert, E. (1993). Symbolic Spatial Reasoning in Object Shapes for Qualitative Reasoning. Spatial Information Theory: Theoretical Basis for GIS. A. U. Frank and I. Campari. Heidelberg-Berlin, Springer Verlag. **716:** 444-463.

Jungert, E. and S. K. Chang (1989). An Algebra for Symbolic Image Manipulation and Transformation. Visual Database Systems. T. L. Kunii, North-Holland**:** 301-317.

Kahmen, H. (1993). Vermessungskunde. Berlin, de Gruyter.

Kemp, K. K. (1993). TUW Offers a New Kind of Course. GIS Europe. **2:** 31.

Kemp, K. K., W. Kuhn, et al. (1993). Making High-Quality GIS Education Accessible: A European Initiative. Geo Info Systems. **3:** 50-52.

Kennedy, H. (1980). Peano Life and Works of Giuseppe Peano, Kluwer.

Kent, W. (1978). Data and Reality - Basic Assumptions in Data Processing Reconsidered. Amsterdam, North-Holland.

Kirschenhofer, P. (1992). Mathematische Grundlagen für GIS, Ausseninstitut der Technischen Universität Wien.

Kirschenhofer, P. (1995). The Mathematical Foundation of Graphs and Topology for GIS. Geographic Information Systems - Material for a Post Graduate Course. A. U. Frank. Vienna, Department of Geoinformation, TU Vienna. **1:** 155-176.

Klein, F. (1872). Vergleichende Betrachtungen über neuere geometrische Forschungen. Erlangen, Verlag Andreas Deichert.

Klein, F., E. R. Hedrick, et al. (2004). Elementary Mathematics from an Advanced Standpoint: Geometry (Dover Books on Mathematics), Dover Publications.

Knuth, D. E. (1992). Axioms and Hulls. Berlin, Germany, Springer-Verlag.

Krantz, D. H., R. D. Luce, et al. (1971). Foundations of Measurement. New York, Academic Press.

Kuhn, W. (1989). Interaktion mit raumbezogenen Informationssystemen - Vom Konstruieren zum Editieren geometrischer Modelle. Zürich, Institut für Geodäsie und Photogrammetrie, ETH Zürich.

Kuipers, B. (1994). Qualitative Reasoning: Modeling and Simulation with Incomplete Knowledge. Cambridge, Mass., The MIT Press.

Lakoff, G. and M. Johnson (1999). Philosophy in the Flesh. New York, Basic Books.

Lakoff, G. and R. E. Núnez (2000). Where Mathematics Comes From - How the Embodied mind Brings Mathematics into Being, Basic Books.

Langran, G. (1989). Time in Geographic Information Systems. Department of Geography. Washington, University of Washington, Seattle, WA.

Langran, G. and N. Chrisman (1988). "A Framework for Temporal Geographic Information." Cartographica **25**(3): 1-14.

Leonardis, A. and H. Bischof (1996). Dealing with Occlusions in the Eigenspace Approach**:** 22.

Lieblich, I. and M. A. Arbib (1982). "Multiple representations of space underlying behavior." The Behavioral and Brain Sciences **5**: 627-659.

Lifschitz, V., Ed. (1990). Formalizing Common Sense - Papers by John McCarthy. Norwood, NJ, Ablex Publishing.

Lifschitz, V. (1990). Understanding Common Sense: John McCarthy's Research in Artificial Intelligence. Formalizing Common Sense - Papers by John McCarthy. V. Lifschitz. Norwood, NJ, Ablex Publishing Company**:** 1-8.

Lindsay, B., M. Stonebraker, et al. (1989). "The Object-Oriented Counter Manifesto."

Lockemann, P. C. and H. C. Mayr (1978). Rechnergestützte Informationssysteme. Berlin, Springer-Verlag.

Loeckx, J., H.-D. Ehrich, et al. (1996). Specification of Abstract Data Types. Chichester, UK and Stuttgart, John Wiley and B.G. Teubner.

Mac Lane, S. and G. Birkhoff (1991). Algebra Third Edition. Providence, Rhode Island, AMS Chelsea Publishing.

MacEachren, A. M. (1995). How Maps Work - Representation, Visualization and Design. New York, Guilford Press.

Maguire, D. J., M. F. Goodchild, et al., Eds. (1991). Geographic Information Systems: Principles and Applications. London, Longman.

Mandelbrot, B. B. (1977). The Fractal Geometry of Nature. New York, W.H. Freeman & Co.

Marble, D. (1984). Proceedings First International Symposium on Spatial Data Handling. Zurich, Switzerland.

Mark, D. (1997, August 29 1997). "The GIS History Project." Retrieved 10.08., 2004, from http://www.geog.buffalo.edu/ncgia/gishist/other_sites.html.

Mark, D. M. and M. J. Egenhofer (1992). An Evaluation of the 9-Intersection for Region-Line Relations. GIS/LIS '92 Proceedings, San Jose, ACSM-ASPRS-URISA-AM/FM.

Mark, D. M. and A. U. Frank, Eds. (1991). Cognitive and Linguistic Aspects of Geographic Space. NATO ASI Series D. Dordrecht, The Netherlands, Kluwer Academic Publishers.

Martin Breunig, Can Türker, et al. (2003). Architectures and Implementations of Spatio-temporal Database Management Systems. Spatio-Temporal Databases: The CHOROCHRONOS Approach. Berlin, Springer**:** 263 - 318.

McCarthy, J. (1985). Epistomological Problems of Artificial Intelligence. Readings in Knowledge Representation. R. J. Brachman and H. J. Levesque. Los Altos, CA, Morgan Kaufman Publishers**:** 24 - 30.

McCarthy, J. (1996, 24.3.1996). "Notes on Formalizing Context." from http://www-formal.stanford.edu/jmc/context3/context3.html.

McCarthy, J. and P. J. Hayes (1969). Some Philosophical Problems from the Standpoint of Artificial Intelligence. Machine Intelligence 4. B. Meltzer and D. Michie. Edinburgh, Edinburgh University Press**:** 463-502.

McCoy, N. H. and T. R. Berger (1977). Algebra: Groups, Rings and other Topics. London, Allyn and Bacon.

McHarg, I. (1969). Design with Nature, Natural History Press.

McHarg, I. L. (1992). Design with Nature. N. Y., USA, Natural History Press.

McKeown, D. J. and R. C. T. Lai (1987). Integrating multiple data representations for spatial databases. Auto-Carto 8, Baltimore, MA, ASPRS & ACSM.

Medak, D. (1997). Lifestyles - A Formal Model. Chorochronos Intensive Workshop '97, Petronell-Carnuntum, Austria, Dept. of Geoinformation, TU Vienna.

Medak, D. (1999). Lifestyles - A Paradigm for the Description of Spatiotemporal Databases. Department of Geoinformation. Vienna, Technical University Vienna.

Medak, D. (2001). Lifestyles. Life and Motion of Socio-Economic Units. A. U. Frank, J. Raper and J.-P. Cheylan. London, Taylor & Francis**:** 140-153.

Messmer, W. (1984). "Wie Basel vermessen wird." Vermessung, Photogrammetrie, Kulturtechnik **4**: 97 -106.

Meyer, B. (1988). Object-Oriented Software Construction. New York, NY, Prentice Hall.

Miller, C. L. (1963). Man-Machine Communications in Civil Engineering, MIT Dept. of Civil Engineering.

Minsky, M. (1985). The Society of Mind. New York, Simon & Schuster.

Molenaar, M. (1995). Spatial Concepts as Implemented in GIS. Geographic Information Systems - Materials for a Post-Graduate Course. A. U. Frank, Department of Geoinformation, TU Vienna**:** 91-154.

Molenaar, M. (1998). An Introduction to the Theory of Spatial Object Modelling for GIS. London, Taylor & Francis.

Montello, D. R. (1993). Scale and Multiple Psychologies of Space. Spatial Information Theory: A Theoretical Basis for GIS. A. U. Frank and I. Campari. Heidelberg-Berlin, Springer Verlag. **716:** 312-321.

NCGIA (1989). "The Research Plan of the National Center for Geographic Information and Analysis." International Journal of Geographical Information Systems **3**(2): 117 - 136.

NCGIA (1989). "The U.S. National Center for Geographic Information and Analysis: An Overview of the Agenda for Research and Education." IJGIS **2**(3): 117-136.

Neumann, H.-G. (1978). Die historische Entwicklung der Datenverarbeitung im Vermessungswesen. Landinformationssysteme Symposium der FIG, Darmstadt, THD Schriftreihe Wissenschaft und Technik 11.

Neumann von, J. and O. Morgenstern (1944). Theory of Games and Economic Behavior. Princeton, NJ, Princeton University Press.

Newman, W. M. and R. F. Sproull (1981). Principles of Interactive Computer Graphics, McGraw - Hill.

OGC. (2000). "The Open GIS Consortium Web Page." Retrieved 21 November, 2000, from http://www.opengis.org.

Openshaw, S. and S. Alvanides (2001). Designing Zoning Systems for the Representation of Socio-Economic Data. Life and Motion of Socio-Economic Units. A. U. Frank, J. Raper and J.-P. Cheylan. London, Taylor & Francis.

Oxley, J. G. (1992). Matroid Theory. Oxford, Oxford University Press.

Parnas, D. L. (1972). "A Technique for Software Module Specification with Examples." ACM Communications **15**(5): 330-336.

Peterson, J., K. Hammond, et al. (1997). "The Haskell 1.4 Report." from http://www.haskell.org/report/index.html.

Peyton Jones, S., J. Hughes, et al. (1999). "Haskell 98: A Non-Strict, Purely Functional Language." from http://www.haskell.org/onlinereport/.

Pierce, B. C. (1993). Basic Category Theory for Computer Scientists. Cambridge, Mass., MIT Press.

Pitt, D. (1985). Categories. Category Theory and Computer Programming; Tutorial and Workshop Proceedings, Springer.

Pontikakis, E. and A. U. Frank (2004). Basic Spatial Data According to User's Needs-Aspects of Data Quality. ISSDQ, Bruck a.d. Leitha, Austria, Department of Geoinformation and Cartography.

Quattrochi, D. A. and M. F. Goodchild, Eds. (1997). Scale in Remote Sensing and GIS. Boca Raton, FL, CRC Press.

Randell, D. A., Z. Cui, et al. (1992). A Spatial Logic Based on Regions and Connection. Third International Conference on the Principles of Knowledge Representation and Reasoning, Los Altos, CA: Morgan-Kaufmann.

Reinhardt, F. and H. Soeder (1991). dtv-Atlas zur Mathematik: Grundlagen, Algebra und Geometrie (Band 1). Muenchen, dtv.

Reiter, R. (1984). Towards a Logical Reconstruction of Relational Database Theory. On Conceptual Modelling, Perspectives from Artificial Intelligence, Databases, and Programming Languages. M. L. Brodie, M. Mylopolous and L. Schmidt. New York, Springer Verlag**:** 191-233.

Reiter, R. (in preparation). Knowledge in Action: Logical Foundations for Describing and Implementing Dynamical Systems.

Reuter, A. (1981). Fehlerbehandlung in Datenbanksystemen. München, Carl Hanser Verlag.

Rhind, D. (1971). "The Production of a Multi-Colour Geological Map by Automated Means." Nachr. aus den Karten und Vermessungswesen(52): 47-51.

Rhind, D. (1991). Environmental Monitoring and Prediction. Handling Geographical Information. I. Masser and M. Blakemore. Essex, Longman Scientific & Technical. **1:** 122-147.

Rhind, D. W. (1991). Counting the People: The Role of GIS. Geographical Information Systems: Principles and Applications. D. J. Maguire, M. F. Goodchild and D. W. Rhind. Essex, Longman Scientific & Technical. **2:** 127-137.

Rosch, E. (1973). On the Internal Structure of Perceptual and Semantic Categories. Cognitive Development and the Acquisition of Language. T. E. Moore. New York, Academic Press.

Rosch, E. (1978). Principles of Categorization. Cognition and Categorization. E. Rosch and B. B. Lloyd. Hillsdale, NJ, Erlbaum.

Rumbaugh, J., Michael Blacha, et al. (1990). Object-Oriented Modeling and Design. Englewood Cliffs, NJ, Prentice Hall.

Samet, H. (1989). Applications of Spatial Data Structures. Computer Graphics, Image Processing and GIS. Reading, MA, Addison-Wesley Publishing Co.

Samet, H. (1990). Applications of Spatial Data Structures. Computer Graphics, Image Processing and GIS. Reading, MASS, Addison-Wesley Publishing Co.

Samet, H. (1990). The Design and Analysis of Spatial Data Structures. Reading, MASS., Addison-Wesley Publishing Company.

Schek, H.-J. (1982). "Remark on the Algebra of Non First Normal Form Relations."

Schek, H.-J. (1985). Towards a Basic Relationsal NF2 Algebra Processor. Second International Conference on Foundations of Data Organization and Algorithms.

Schek, H.-J. and M. H. Scholl (1983). Die NF2-Relationenalgebra zur Einheitlichen Manipulation Externer, Konzeptueller und Interner Datenstrukturen, Springer-Verlag Berlin Heidelberg New York Tokyo.

Schneider, M. (1997). Spatial Data Types for Database Systems. Berlin-Heidelberg, Springer-Verlag.

Schönenberger, W. (1976). Schweizerisches Zivilgesetzbuch. Zürich, Schulthess Polygraphischer Verlag AG.

Schröder, E. (1890). Vorlesungen über die Algebra der Logik (Exakte Logik). Leipzig, Teubner.

Sellis, T. and M. Koubarakis, Eds. (2003). Spatio-Temporal Databases. Berlin Heidelberg, Springer-Verlag.

Sernadas, A. (1980). "Temporal Aspects of Logical Procedure Definition." Information Systems **5**: 167-187.

Sester, M. (1996). "Acquisition of Rules for the Transition between Multiple Representations in a GIS." International Archives of Photogrammetry and Remote Sensing **XXXI**(B4): 768-773.

Shannon, C. E. (1938). "A Symbolic Analysis of Relay and Switching Circuits." AIEE Trans. **57**: 713 - 723.

Shannon, C. E. and W. Weaver (1949). The Mathematical Theory of Communication. Urbana, Illinois, The University of Illinois Press.

Shariff, R., M. Egenhofer, et al. (1998). "Natural-Language Spatial Relations between Linear and Areal Objects: The Topology and Metric of English-Language Terms." IJGIS **12**(3): 215-246.

Shi, W., P. F. Fisher, et al. (2002). Spatial Data Quality, Taylor & Francis.

Shipman, D. W. (1981). "The Functional Data Model and the Data Language DAPLEX." ACM Transactions on Database Systems **6**(March).

Sinowjew, A. A. (1968). Über mehrwertige Logik. Berlin, Deutscher Verlag der Wissenschaften.

Sinton, D. (1978). The Inherent Structure of Information as a Constraint to Analysis: Mapped Thematic Data as a Case Study. Harvard Papers on GIS. G. Dutton. Reading, Mass., Addison-Wesley. **Vol.7**.

Sowa, J. F. (1998). Knowledge Representation: Logical, Philosophical, and Computational Foundations. Boston, PWS Publishing.

Steiner, D. and H. Gilgen (1984). Relational Modelling as a Design and Evaluation Technique for Interactive Geographical Data Processing. International Symposium on Spatial Data Handing, Zurich, Switzerland, Geographisches Institut, Abteilung Kartographie/EDV.

Stevens, S. S. (1946). "On the Theory of Scales of Measurement." Science **103**(2684): 677-680.

Stolfi, J. (1991). Oriented Projective Geometry. San Diego, CA, USA, Academic Press Professional, Inc.

Stonebraker, M., L. A. Rowe, et al. (1990). Third-generation Data Base System Manifesto, UC Berkeley: Electronics Research Lab.

Stonebreaker, M. (1993). The SEQUOIA 2000 Project. Advances in Spatial Databases. D. Abel and B. C. Ooi. Heidelberg, Springer Verla**:** 397 - 412.

Stroustrup, B. (1986). The C++ Programming Language. Reading, Mass., Addison-Wesley Publishing Company.

Stroustrup, B. (1991). The C++ Programming Language. Reading, Mass., Addison-Wesley.

Tansel, A. U., J. Clifford, et al. (1993). Temporal Databases. Redwood City, CA, Benjamin Cummings.

Tarski, A. (1941). "On the Calculus of Relations." The Journal of Symbolic Logic **6**(3): 73-89.

Tarski, A. (1977). Einführung in die mathematische Logik. Göttingen, Vandenhoeck & Ruprecht.

Timpf, S. (1998). Hierarchical Structures in Map Series. Faculty of Science and Technology. Vienna, Technical University Vienna**:** 124.

Timpf, S. and T. Devogele (1997). New Tools for Multiple Representations. ICC'97, Stockholm, Editor: Lars Ottoson.

Timpf, S. and A. U. Frank (1997). Using Hierarchical Spatial Data Structures for Hierarchical Spatial Reasoning. Spatial Information Theory - A Theoretical Basis for GIS (International Conference COSIT'97). S. C. Hirtle and A. U. Frank. Berlin, Springer-Verlag. **Lecture Notes in Computer Science 1329:** 69-83.

Timpf, S., G. S. Volta, et al. (1992). A Conceptual Model of Wayfinding Using Multiple Levels of Abstractions. Theories and Methods of Spatio-Temporal Reasoning in Geographic Space. A. U. Frank, I. Campari and U. Formentini. Heidelberg-Berlin, Springer Verlag. **639:** 348-367.

Tinkler, K. J. (1988). Nystuen/Dacey Nodal Analysis (Monograph / Institute of Mathematical Geography), Michigan Document Services.

Tobler, W. (1961). Map Transformations of Geographic Space. Seattle, Washington, Uinversity of Washington.

Tobler, W. (1992). "Preliminary Representation of World Population by Spherical Harmonics." Proc. Nail. Acad. Sc. **89**: 6262-6264.

Tobler, W. R. (198?). "Resolution, Resampling, and all that." 129-137.

Tobler, W. R. and S. Wineberg (1971). "A Cappadocian Speculation." Nature **231**(May 7): 39-42.

Tomlin, C. D. (1983). Digital Cartographic Modeling Techniques in Environmental Planning, Yale Graduate School, Division of Forestry and Environmental Studies.

Tomlin, C. D. (1990). Geographic Information Systems and Cartographic Modeling. New York, Prentice Hall.

Tomlin, C. D. (1991). Cartographic Modelling. Geographical Information Systems: Principles and Applications. D. J. Maguire, M. F. Goodchild and D. W. Rhind. Essex, Longman Scientific & Technical. **1:** 361-374.

Tomlin, D. (1994). "Map Algebra: One Perspective." Landscape and Urban Planning **30**: 3-12.

Tomlinson, R. F. (1984). Geographic Information Systems - A New Frontier. International Symposium on Spatial Data Handling, Zurich, Switzerland.

Tomlinson, R. F., H. W. Calkins, et al. (1976). Computer Handling of Geographical Data: An Examination of Selected Geographic Information Systems. Paris, The Unesco Press.

Tufte, E. R. (1997). Visual Explanations - Images, Quantities, Evidence and Narrative. Ceshire, Connecticut, Graphics Press.

Ullman, J. D. (1982). Principles of Database Systems. Rockville, MD, Computer Science Press.

Unwin, D. J. (1990). "A Syllabus for Teaching Geographical Information Systems." International Journal of Geographical Information Systems **4**(4): 457-465.

van Benthem, J. F. A. K. (1983). The Logic of Time, Reidel Publ. Comp.

Vckovski, A. (1998). Interoperable and Distributed Processing in GIS. London, Taylor & Francis.

Vckovski, A. and F. Bucher. (1998, Dec. 1995). "Virtual Data Sets - Smart Data for Environmental Applications." 1998, from http://www.ncgia.ucsb.edu/conf/SANTA_FE_CD-ROM/sf_papers/vckovski_andrej/vbpaper.htm.

Vetter, M. (1977). Principles of Data Base Systems. International Computing Symposium, Liege, Belgium.

Wadler, P. (1989). Theorems for Free! Functional Programming Languages and Computer Architecture, ACM**:** 347-359.

Walters, R. F. C. (1991). Categories and Computer Science. Cambridge, UK, Carslaw Publications.

Wegner, P. (1987). Dimensions of Object-Based Language Design. Object Oriented Programming Systems, Languages and Applications (OOPSLA'87), Orlando, Florida; October 1987.

White, M. S. (1979). A Survey of the Mathematics of Maps. Auto Carto IV.

White, M. S. and P. E. Griffin (1979). Coordinate Free Cartography. Auto Carto IV.

Whitehead, A. (1898). A Treatise on Universal Algebra. Cambridge, Cambridge University Press.

Whitehead, A. and B. Russell (1910-1913). Principia Mathematica. Cambridge, Cambridge University Press.

Wirth, N. (1971). "The Programming Language Pascal." Acta Informatica **1**(1): 35-63.

Wittgenstein, L. (1960). Tractatus logico-philosophicus. London, Routledge & Kegan Paul.

Wolfram, S. (1988). Mathematica. Bonn, Addison-Wesley.

Wolfram, S. (2002). A New Kind of Science, Wolfram Media.

Zadeh, L. A. (1974). "Fuzzy Logic and Its Application to Approximate Reasoning." Information Processing.

Zaniolo, C., P. C. Lockemann, et al., Eds. (2000). Advances in Database Technology - Proceedings of EDBT 2000 (Konstanz, Germany, March 2000). Lecture Notes in Computer Science. Berlin Heidelberg, Springer-Verlag.

Zehnder, C. A. (1998). Informationssysteme und Datenbanken. Stuttgart, B. G. Teubner.

INDEX

D