WEIPING YANG

The Design of a Dynamic Voronoi Map Object (VMO) Model for Sustainable Forestry Data Management

Thèse
présentée
à la Faculté des études supérieures
de l'Université Laval
pour l'obtention
du grade de Philosophiae Doctor (Ph.D.)

Département des sciences géomatiques
FACULTÉ DE FORESTERIE ET DE GÉOMATIQUE
UNIVERSITÉ LAVAL
QUÉBEC

Novembre 1998

The unleashed power of the atom has changed everything, save the way we think and thus we drift toward unparalleled catastrophe.

Albert Einstein

Résumé

Le diagramme de Voronoi est une structure géométrique puissante et attrayante pour un grand nombre d'applications. La présente thèse étudie la souplesse d'une telle structure géométrique, appelé le diagramme Voronoi dynamique de points et de segments de lignes, appliquée aux systèmes d'information géographique (SIG). En particulier, la question qui concerne cette thèse est : Étant donné le domaine d'application qui est la foresterie, est-ce que le diagramme Voronoi dynamique est un modèle de données utile pour apporter un support aux applications concernées par le développement durable en foresterie?

Cette thèse apporte une réponse fortement positive à la question précédente. En révisant les caractéristiques forestières du développement durable, cette thèse résume les nécessités correspondant à la validation des modèles de données spatiales. Après l'examen des modèles de données spatiales traditionnels utilisés dans la plupart des systèmes SIG courants, cette thèse est en accord avec le fait que le modèle dynamique Voronoi peut supporter l'intégration de la topographie et de la géométrie. Il satisfait aussi à toutes les exigences d'un SIG dynamique. Cette discussion est soutenue par une incorporation du SIG dynamique Voronoi dans sa forme primitive et en présentant les opérations habituelles des SIG à travers le modèle de données Voronoi.

Cette thèse contribue au développement d'un SIG Voronoi dynamique en proposant un modèle Voronoi Map Object (VMO) qui soustrait les limitations associées à l'occupation de la mémoire pour les diagrammes Voronoi de grande taille. Le modèle VMO est réalisé en sous-divisant le diagramme Voronoi en sous-diagrammes et en les représentants avec une structure d'objet hiérarchique. Chaque nœud sur la structure est un VMO et supporte entièrement la topologie et la géométrie reliées de même que les opérations sur les objets. Cette thèse décrit l'algorithme utilisé pour sous-diviser (et coller) un diagramme Voronoi et également le formalisme associé au modèle VMO.

Finalement, la thèse discute de la conception d'un système de gestion des données forestières utilisant le modèle VMO. La discussion couvre le modèle objet, le modèle dynamique, le modèle fonctionnel et l'architecture logiciel du système. L'application du modèle VMO pour le traitement parallèle de problèmes spatiaux de même que pour la généralisation automatisée de cartes sont également brièvement discutées.

Abstract

The question concerned in the thesis is: Given forestry as an application domain, is the dynamic Voronoi diagram a useful GIS data model to support applications concerned with the sustainability of forestry?

The thesis provides a strong positive answer to the above question. After examining traditional spatial data models and data structures used in current GIS, the thesis argues that the dynamic Voronoi data model can support integration of the topology and geometry and satisfies all the requirements for a dynamic GIS.

The thesis contributes to the development of a dynamic Voronoi GIS by proposing a Voronoi Map Object (VMO) model. The VMO model is achieved by partitioning the Voronoi diagram into subdiagrams and by representing them with a hierarchical object structure.

Résumé

La question qui concerne cette thèse est: Étant donné le domaine d'application qui est la foresterie, est-ce que le diagramme Voronoi dynamique est un modèle de données utile pour apporter un support aux applications concernées par le développement durable en foresterie?

Cette thèse apporte une réponse fortement positive à la question précédente. Après l'examen des modèles de données spatiales traditionnels utilisés dans les SIG courants, cette thèse est en accord avec le fait que le modèle dyamique Voronoi peut supporter l'intégration de la topographie et de la géométrie.

Cette thèse contribue au développement d'un SIG Voronoi dynamique en proposant un modèle Voronoi Map Object (VMO). Le modèle VMO est réalisé en sous-divisant le diagramme Voronoi en sous-diagrammes et en les représentants avec une structure d'objet hiérarchique.

C. Gold.
Director

Student

Abstract

The Voronoi diagram is a powerful geometric structure attractive to many applications, especially when it is developed with computers. This thesis investigates the flexibility of such a geometric structure, called the dynamic Voronoi diagram of points and line segments, applied to GIS, geographical information systems. In particular, the question concerned in the thesis is: Given forestry as an application domain, is the dynamic Voronoi diagram a useful GIS data model to support applications concerned with the sustainability of forestry?

The thesis provides a strong positive answer to the above question. By reviewing the characteristics of sustainable forestry, the thesis summarizes the corresponding requirements for the supporting spatial data models. After examining traditional spatial data models and data structures used in current GIS systems, the thesis argues that the dynamic Voronoi data model can support integration of the topology and geometry and satisfies all the requirements for a dynamic GIS. The argument is supported by a computer implementation of the dynamic Voronoi GIS in its primitive form and by presenting common GIS operations over the Voronoi data model.

The thesis contributes to the development of a dynamic Voronoi GIS by proposing a Voronoi Map Object (VMO) model which removes the limitation of memory occupation for large Voronoi diagrams. The VMO model is achieved by partitioning the Voronoi diagram into subdiagrams and by representing them with a hierarchical object structure. Each node on the structure is a VMO and support full topology and geometry about, and operations on the object. The thesis describes the algorithm for partitioning (and pasting) a Voronoi diagram and the formalism of the VMO model.

Finally, the thesis discusses design issues for a forestry data management system using the VMO model. The discussion covers the object model, the dynamic model, the functional model, and the software architecture of the system. The applications of the VMO model for parallel processing of spatial problems and for automated map generalization are also briefly discussed.

Contents

Figures

Tables

Acknowledgements

Many people helped to make this thesis possible. First, I am indebted to Dr. Christopher Gold, my thesis supervisor. It was Chris who inspired my interest in the study of the Voronoi diagram and generously allowed numerous opportunities for me to share his experience in this subject. His humor, encouragement, support, and understanding made the time working with him worthwhile and very much memorable. Thank you indeed, Chris.

I would like to thank Dr. Geoffrey Edwards of the Département des sciences géomatiques at Université Laval, and Dr. Michael Worboys of the Department of Computer Science at Keele University, UK, for their careful reading through and correcting the thesis. Through personal discussions and his wonderful book, Mike helped me with some aspects of mathematics which led to the improvement of the presentation of the thesis. Geoffrey encouraged me in many ways during my study and work at Université Laval. His comments and suggestions made a lot of the statements in the thesis clearer. I thank Dr. Jean-Marie Beaulieu of the Département d'informatique, Université Laval, for his constructive comments and suggestions. Dr. Yvan Bédard, of the Département des sciences géomatiques at Université Laval, pointed out mistakes and weakness of the thesis in a variety of places and advised on how to strengthen them. I appreciate his efforts in providing me with valuable reading materials which extended the scope of the discussions.

The consistent financial support from the Natural Sciences and Engineering Research Council of Canada (NSERC) and the Association des industries forestières du Québec (AIFQ) is gratefully acknowledged, without which the undertaking of the research would have been very difficult.

I would also like to take the opportunity to thank many of my colleagues and fellow students in the Centre de recherche en géomatique at Université Laval for their understanding, help, and friendship.

Special thanks should go to Mrs. Valerie Gold for her help with grammar checking and correcting for the thesis, and especially, for her encouragement and moral support.

The thesis would never have been finished without the unconditional support, tolerance, and love from my wife, Yunzhu, and my daughter, Theresa. I am always indebted to and proud of them. I would also like to express my thanks to my parents and other family members for their help and moral support during my difficult time.

Chapter 1

Introduction

Forests constitute an imperative semi-natural resource for the existence of a global society. The significant role forests play in the global ecosystem cannot be over-estimated. Unfortunately, rapid deforestation has been occurring world wide, especially in developing countries, due to the following factors: clearing the forest land for farming, the demand for firewood and fodder, excessive commercial logging for short-term economical profits, and fire – intentional and natural. These factors are aggravated by population growth, infrastructure and industrial development, and bad planning and management. The cost of deforestation to society is tremendous. Beside obvious and presently hard-to-measure economic losses, some of the earth's plant and animal species are in danger of extinction, and, deforestation has resulted in the recent increase in the atmospheric concentration of carbon dioxide which leads to increased global warming [World Bank 1993].

1.1 Sustainable Forest Development

In order to achieve a biologically and economically sound society, we have to take seriously the requirements for sustained forest management to meet the needs of the present generation without compromising the needs of future generations [World Commission on Environment and Development 1987]. The word "sustainability" in forestry refers to the application of sustained-yield management practices to forests in order to ensure a continuous flow of desired forest products and services, without undue reduction of their inherent value and future productivity, and without undue undesirable effects on the physical and social environment [ITTO 1992].

Sustainably developing forests needs an integrated management plan, in addition to many other measures such as institutional, policy, and cultural changes. Integrated planning and management of forest resources means that the application domain should not be considered in isolation of its neighbouring and overlapping domains. Instead, forest planning and management should be practiced by treating forest resources as an indispensable subsystem of the ecosystem, including: soil, water, biological diversity, genetic diversity, landscape patterns, and cultural evolution -- all are critical components to maintain ecosystem integrity. Ecosystems are communities of organisms working together with their environment as integrated units. They are places where all plants, animals, soils, waters, climate, people, and processes of life interact as a whole. These ecosystems/places may be small, such as a rotting log, or large, such as a continent or the biosphere. The smaller ecosystems are subsets of the larger ecosystems; that is, a pond is a subset of a watershed, which is a subset of a landscape, and so forth. All ecosystems have flows of things -- organisms, energy, water, air, and nutrients -- moving among them. And all ecosystems change over space and time. Therefore, it is not possible to draw a line around an ecosystem and mandate that it stay the same or stay in place for all time. Managing ecosystems means working with the processes that cause them to vary and to change [Salwasser et al. 1993].

Forest planning and management should be based on an ecosystem approach with the following principle: to conserve biologically and genetically diverse and productive landscapes within local, regional, national, and global contexts. The ecosystem approach embodies three fundamental concepts: designating the physical boundary of the system and its components, understanding the interaction of its parts as a functioning whole, and understanding the relation between the system and its context. Context in this sense means both the external factors that influence the system as well as the internal information that must be synthesized at the scale of the defined system if we are to have any understanding of it. For a continental ecosystem, global air pollution and population growth are examples of external context and local politics and endangered species are examples of internal context [Maser 1994].

1.2 Characteristics of Sustainable Forest Ecosystem Management

Although understood in principle, a scientifically precise definition of what constitutes an appropriate sustainable forest ecosystem management practice is still missing. This is because of the lack of knowledge of the natural functioning and response of forest ecosystems [Sample 1993]. Nevertheless, several basic characteristics of sustainable forest management have been identified through recent research and practices in ecosystem management:

It should operate across large spatial scale. The system should not only consider a broad array of species within the management unit but also a broad concept of the management unit itself -- from focusing on forestland parcels and stands to focusing on landscape/regional-scale areas defined along ecological boundaries. Real sustainability must account for even larger scales up to and including the global environment. In most cases, landscape areas encompass a host of both public and private forestlands. It is clear that a far higher level of cooperation, coordination, and collaboration would be needed among the various public and private landowners and managers occupying a set of ecologically defined management units [Franklin 1989].

This perspective strongly suggests that the management system should have a hierarchical structure, from the lowest level of productive and operational scale to the highest level of directional policy steering and control scale. The hierarchical structure is not only institutional (organizations and people involved), but also needs to be implementable with available technology. To support integrated solutions, the forest ecosystem management needs to run more or less the same set of data, implying that the classification of data should also follow the correspondent hierarchical structure, from the finest to the coarsest resolution.

Sustainable forest management needs to be highly open and cooperative. Institutional and policy changes must be addressed to facilitate closer cooperation, coordination, and

collaboration among adjacent public and private landowners in the establishment and achievement of ecosystem management goals. In addition, function-based, target-oriented resource management hierarchies should be developed within a more open environment to be conducive to multidisciplinary approaches. The open environment should encourage dissimulation of philosophies toward sustainable forest ecosystem development, scientific exchanges, and most importantly, sharing of data. Barriers to the exchange of environmental data across different institutions should be minimized to allow multiuse of the data. Standards for data collection, formatting, and accuracy control need to be set up and agreed upon among involved parties. On the other hand, flexibility of the management system to accept, interpret, and integrate data from different sources with different formats would also be highly desirable.

The management system must be able to reconcile overlapping goals. An ecosystem management approach requires that management goals and actions simultaneously satisfy three conditions: ecological viability (environmentally sound), economical feasibility (affordable), and social desirability (politically acceptable) (Figure 1.1). If the balance among these three criteria is not reasonable, there is a high likelihood that the ecosystem will not be sustainable [Zonneveld 1990].

Due to various interests and objectives held by private companies, communities, public citizens, and government agencies, conflicting management goals necessarily exist even before an integrated management system is established and continue through its life-cycle. Negotiations, education, and stipulated regulations would be needed to compromise overlapped goals. The management system should also be prepared with alternatives for the choice of optimized decisions. To achieve this, powerful data analysis functions, matched with sophisticated data manipulation capabilities, would have to be built into the system.

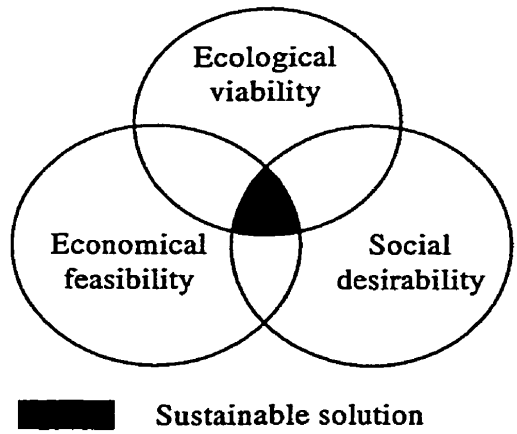


Figure 1.1 Ecosystem solution triangle (Adapted from Zonneveld [1990])

The management system must handle dynamic processes. Forest ecosystem management goes beyond mere description of conditions or states presumed to be static. Emphasis should be on greater understanding of linked processes, the positive and negative feedback between linked processes, and the relationship between processes at different temporal and spatial scales. Linkages between elements (e.g. reciprocal linkage between carbon and nitrogen cycles, influenced by browsing mammals and insects) as they cycle between the trees and soil, impose nonlinear dynamics on forest growth and stand development. Biological dynamic processes can dramatically alter the regional ecosystem at both forest stand (species composition and structure) and landscape (the relations of size and location of various forest ecosystems) scales [Mladenoff and Pastor 1993].

Concerning the requirements for large spatial scales and dynamic processes, managing sustainable forest ecosystems requires viewing management objectives at large spatial as well as temporal scales to accommodate a broad range of objectives. The silviculture tool-box must be enlarged. Simple methods (e.g. even-aged or uneven-aged management) need to be combined with complex ones (e.g. partial cutting, group selection, retaining seed trees and introducing new species, and extended rotation) to meet more complex objectives that promote ecosystem sustainability at both landscape and stand levels. This change would allow the development and application of more creative silviculture techniques that provide for long-term commodity production and the maintenance of biodiversity and long-term

productive potential, thus buffering against climatic changes. The suggested sustainable forest management of silviculture techniques at the stand level and the landscape level (landscape-scale integration) is termed *dynamic landscape heterogeneity* [Mladenoff and Pastor 1993].

The modification of traditional silviculture methods necessarily breaks with classical forest management concepts. In standard forestry, the forest is typically divided into management compartments/parcels that are subdivided into stands, consisting of tens of acres, to which particular silviculture techniques are applied. Parcels and stands are generally considered to be fixed and independent. Applying dynamic landscape heterogeneity in management requires that over longer time frames these boundaries should be considered fluid and that management techniques should be aggregated over larger areas to maintain landscape-scale patterns and processes. In the natural landscape, overlapping fires and other disturbances also cause boundaries to be fluid.

The management system must incorporate a long-term life cycle. Forest ecosystems feature various temporal cycles, due to the complex composition of elements and natural processes within each functional ecosystem. The impacts of linked natural processes and disturbance events (wildfire, windstorm, insect outbreak, and tree diseases), human influences such as harvesting and silviculture actions on site productivity of forestland and sustainability of landscape-regions become evident generally after long time horizons (three rotations as suggested by Comerford et al. [1994]). This suggests that forest ecosystem management should be targeted on a long-term scale. A long-term management scale is practically and strategically desirable from two points of view:

First, from the point of view of forest managers, the sustainable productivity of forest biomass, usually wood, is their major concern. Negative impacts of forest harvesting can be reduced by the appropriate choice of rotation length, harvest season, road construction methods, harvesting equipment, utilization standards (e.g. whole-tree vs. stem-only harvesting) and silviculture system (e.g. clearcutting vs. partial cutting). The site productivity may also be increased, at least in the short-term, by stimulating nutrient cycles,

changing plant structures, adding fertilizers, and introducing improved genetic stock [Morris and Miller 1994]. All these applications should only be used with a basic of knowledge of the historical and current site conditions. It is essential to collect and compile available information on the entire stand to make an informed evaluation. Furthermore, any events and actions in the field should be recorded in the system for future assessment.

Second, from the point of view of forestry researchers, especially forest biologists, studies of dynamic processes and the effect of each process, event, disturbance, and forestry operation on the pyramid of forest ecosystems are long-term projects. For example, current methodologies for studying long-term forest site productivity fall into one of three categories: chronosequential, retrospective, and long-term field trial [Dyck and Cole 1994]. All these methods rely on adequate documentation of historical information or on long-term experiments and observations. More importantly, the results of forest ecosystem research (e.g. various predicting models) need to be validated through calibrating, testing, and comparing (with practical observation, experiments, and forestry operations), sometimes spanning multiple rotations. This imposes a great difficulty because the length of practical experiments may well be beyond the serving periods of researchers and budget limits. A sustainable forest management system should be equipped with the capacity to maintain a minimum set of data which standardizes important environmental and site variables [Comerford et al. 1994]. The structure changes of the data set over time should equally be maintained so that any previous states of the forestland, operations and experiments applied to it can be known if desired, thus benefiting continuous management and research over long periods of time.

The management system should have the capability of simulation. Simulation modelling is particularly useful in forestry because of both the long time scales involved and the structural and functional complexity of forest ecosystems. Computer simulations provide a means to organize and evaluate knowledge and hypotheses of both the structural and functional properties of forest ecosystems. Once validated and calibrated with data sets, a simulation allows us to ask "what if" questions, or to interrogate alternatives of forest management strategies over past, present, and future states of the forest ecosystems.

Simulation models have been used for a wide range of applications including production forecasting, yield control, and the evaluation of alternative management operations [Proe et al. 1994]. Most of these models are based on linear programming, which have demonstrated difficulties in plan implementation. This stems from the fact that there are no spatial representations of the optimal solutions developed with these models [Sample 1995]. Improved planning and predicting models are seriously needed to incorporate spatial and temporal configurations of variables.

The management priorities must be adaptive. Forest ecosystems are dynamic and subject to change. At best, a forest ecosystem is managed based on available knowledge, experience, and hypotheses. Thus management goals are set; inventory information is compiled; and plans and decisions are made and implemented using current technology and limited knowledge. It is important to track progress toward goals. This includes assessing present forest/environmental conditions, using quantifiable indicators, and monitoring changes in those conditions over a long time frame. As updated data are obtained and reviewed, suggestions on improving or adjusting details of management plans may emerge. The system should be "adaptive" to the modification of goals, inventory structures, or adopting updated technologies (Figure 1.2).

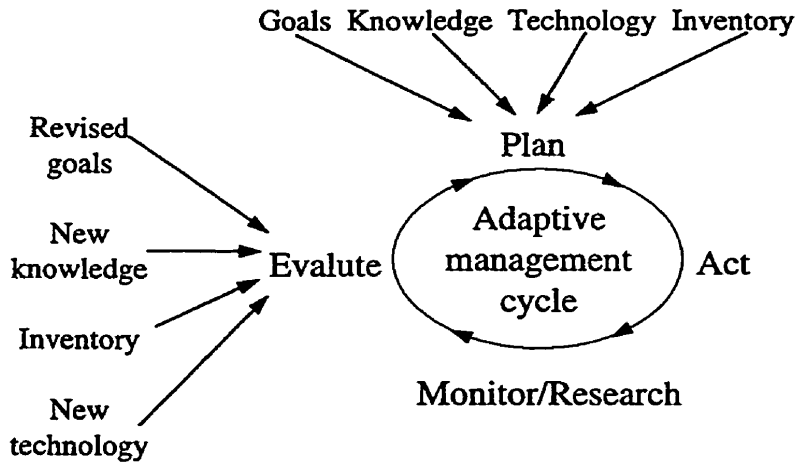


Figure 1.2 The "adaptive" concept of ecosystem management
(Modified from Birch et al. [1993])

In summary, sustainable forest ecosystem management involves multi-disciplinary and integrated solutions. It is not merely a materialized system, but a set of structures and processes requiring philosophical, conceptual, societal, and institutional attendants. For convenience, we call these attendants collectively the *cultural component* of ecosystem management. The other components of the management system include *hardware*, *software*, and *data*. Hardware refers to computers and related devices with which data can be physically stored, displayed, and plotted. Software refers broadly to a set of computer programs which controls operation of the hardware/software, and manipulates data. Data constitutes an important and expensive part of the management contents. The rest of this thesis is actually dedicated to the design of structures to manage data such that they can be manipulated elegantly to meet the requirements of forest ecosystem management.

1.3 Features of GIS for Forestry Management and Decision Support

Environmental concerns, public pressure, and economic growth no longer allow empty discussion on what defines a forest ecosystem and ecosystem management or whether sustainable management can be achieved. Given these pressures, we must achieve a sustainable, reasonably cost effective management system as soon as possible. The call for sustainable forest ecosystem management echoes around the world. In Canada, for example, a National Round Table on the Environment and the Economy was convened to discuss diverse issues related to sustainable forest ecosystems [National Round Table on the Environment and the Economy 1993]. The Canadian vision of sustainable forest ecosystems has been defined. Based on this the Canadian Forest Service (CFS) dictates strategies for developing sustainable forestry [Canadian Forest Service 1994]. Of the five strategies guiding CFS's activities towards the end of this century, the one gaining significant focus is the development of computer-based geographic information and decision support systems. Implementing these systems can integrate biological, environmental, economic and social information over a variety of time frames and spatial scales, and generate forest management alternatives that are understandable and useable by

forest managers. It is realized that sustainable forest development will only become possible when we are able to integrate, manipulate and interpret complex data sets from many different sources, over wide temporal and geographic scales.

The geographical information systems (GIS) for forestry management and spatial decision support systems (SDSS) must meet the requirements of sustainable forestry development. In what follows, we attempt to identify important operational features of such a GIS based on the characteristics of sustainable forestry management. Definitions of GIS and SDSS, and their hardware and software components are presented in Appendix A.

- The GIS needs to have a spatial data model based on which a geographical database can be constructed. The spatial data model should not only support cartographic operations on the spatial objects in the database, but also allow versatile spatial analyses that require topological relationships of the spatial objects. The presence of a topologically enabled spatial data model in the GIS is fundamental to support the requirements for a sustainable forestry management system.
- The GIS needs to support spatial concepts such as country, region, forest territory and stand, forest fire, watersheds, road, GPS (Global Positioning Systems) survey points, population density, income rate, etc. Some of the spatial concepts are by nature hierarchical, for example, a country has provinces and counties at its lower levels. Depending on the spatial concepts used, the objects used to describe these concepts may overlap each other geographically. For example, a watershed can overlap a few forest territories. It should be possible to find locations of entities from the spatial database either by a hierarchy or by an overlapping concept. By hierarchy, a search from a higher level entity to lower ones reveals more detail about the entity. By overlapping, entities cross-referenced in different hierarchical concepts may be required to satisfy a query.
- The GIS needs to act as an integrator (or a warehouse) of a network of geographical databases, possibly heterogeneous and geographically distributed. In the integration, a client GIS can maintain, in its database, some content information (metadata) of geographical entities without storing their detailed data. Instead, linkages to servers where the detailed data, as well as query and analytical services are provided. In the

case that a data server does not support a specific analysis or operation, the detailed data can be transmitted to the client or other GIS in the network. This feature requires OLAP (Online Analytical Processing) capability similar to common MIS (Management Information Systems). The difference is that the client GIS has the spatial data model support to manage, manipulate, and maintain spatial metadata, and if necessary, detail spatial objects referred by the metadata. A client GIS can also act as a server to other clients for whom spatial query and analysis are performed.

- The GIS needs to be dynamic in that updates to the geographical database should be incorporated through short transactions without excessive operation or long locks to the database. Besides, any modification to spatial objects of the database should affect only the spatial objects involved. Services being provided by unaffected spatial objects in the database should not be discontinued by the modification. This feature is important to OLAP applications and is advantageous to what-if questions and simulations.
- The GIS needs to have a component architecture such that any component of the system can be integrated into existing corporation information and decision systems without extensive technical efforts. The spatial components, which are geometrically and topologically enabled spatial objects, should provide common GIS functions and allow themselves to be aggregated with other spatial components. With the component architecture, multi-media representations of geographical entities can be implemented.
- The GIS needs to support dynamic versioning of spatial components. Changes to spatial objects should be tractable given a temporal scale. This feature supports undo-changes to spatial objects, and simulations demonstrating evolutions of spatial objects.

1.4 The Problem and Objective of the Research

The primary objective of the thesis is to design an advanced spatial data model, which can be used to manage and manipulate dynamic spatial objects in a GIS. The core technology of the GIS is the spatial data model. Without a capable spatial data model, the support of the desired features would be difficult. The rigorous design needs to consider not only the theoretic foundation of the data model, but also its practical implementation with

computing software. The advancement of the data model needs to be justified by theoretical analysis and practice.

The data model needs to be studied under the context of geographical information processing. This implies that it should be able to appropriately treat special characteristics of geographical data. Large in quantity, irregular in distribution and density, and volatile with dynamic changes are among the characteristics. One of the targeted application scenarios of the data model is for the development of a SDSS for forest harvesting management. The management system requires that the hardware, software, data, as well as data management will be distributed at distant sites but the system still functions as an integrated whole. Data should be represented and managed hierarchically with appropriate scales. Data should be accessible simultaneously by multi-users and modifiable in a dynamic fashion. The history of data evolution needs to be preserved and managed. In addition, the management system should encourage multimedia presentation of data. Finally, the whole data management system should have an open architecture, to be adaptive to changes.

The objective is determined from two aspects. On the one hand, published documents suggest that most current GIS systems are usually developed based on a *two-step* process for managing spatial data. The first step (map creation) involves a geometric data model such as the R-tree, which organizes digital spatial objects such that they are assigned addresses and therefore can be easily accessed. The second step (spatial analysis), if required, involves building topological relationships among spatial objects, taking advantages of the result from the first step. Examples of GIS equipped with the two-step architecture include Arc/Info™ and Intergraph MGE/Dynamo™. With software technology, these two steps may be streamlined now without being noticed by users. The theoretical basis stays the same. There are still two kinds of spatial data models, geometrical and topological, built in the system. MapInfo™ desktop GIS is an example that is not equipped with the topological process. The separation of geometrical and topological data models in a fully equipped GIS presents many disadvantages. Analysing the spatial

data architecture used by prevailing GIS, and finding out the shortcomings of the two-model system is the first task of the thesis.

On the other hand, the Voronoi data model does not follow the two-step approach to process geometry and topology of spatial data. The research and development on the dynamic Voronoi diagrams has been carried out for a few years in the Industrial Chair in Geomatics Applied to Forestry, Centre de recherche en geomatique at Université Laval. The data structures and algorithms are fully published. The results have demonstrated advantages of using the dynamic Voronoi diagram as the integrated GIS data model.

The main problems, however, are that the Voronoi diagrams, being complicated and detailed, must be presented at a single level and stay in main memory, and that it is difficult to divide a Voronoi diagram into spatial pages. These difficulties prevent the dynamic Voronoi diagram from being fully employed in a GIS environment, where large data sets are often involved. Investigating effective methods to overcome these difficulties therefore becomes the most important task of the thesis research.

The decision to target at the forestry application domain, using the scenario of forestry data management, is made as one of the important liaisons between the Industrial Chair in Geomatics Applied to Forestry and the Association des Industries Foresterie du Quebec (AIFQ). It is hoped that the research and development carried out in this thesis could be useful to the development of an advanced forestry data management system. The thesis will outline the design of the system components for a forestry spatial data management, applying the data model to be developed. Due to the major endeavour towards developing a sound, integrated spatial data model, the effort in the systems design will be limited.

1.5 Methodology

- Examine the forestry needs for decision support for harvest management. Among the numerous issues involved in forest harvesting, the examination will focus especially on forest harvesting data management.
- Evaluate the weaknesses of traditional GIS for decision support. The evaluation will emphasize on the technical aspects of a GIS with regard to dynamic data updates.
- Suggest improved GIS data models and algorithms to respond to these weaknesses.
- Implement and test these methods at the technical level. The research in this part will explore deeply into the technical complexity, if any, of the proposed GIS data model.
- Evaluate the usefulness of the proposed GIS data model in forest harvesting. The evaluation will consider the integration of the improved GIS data model and algorithms into the prototype of a forest data management system.

1.5.1 The Approach

It is found that forest harvest management has needs for dynamic features not always available in a traditional GIS. In particular, the dynamic features concern the "local updating" of forest digital maps, required both for the addition of annual information, such as the previous year's cutting, and for the experiment in a "what-if" fashion, with suggested future interventions. The updates need to be local in the sense that they will not have to be propagated to the whole spatial database of the GIS, in order to maintain the integrity of the spatial relationships.

These dynamic features are primarily the ability to add and delete lines that form polygon boundaries, without the necessity of expensively re-structuring the "topology" of the map sheet holding the polygons. Given this ability, map updating and what-if queries performed by a GIS would become much simplified and realistic for real-time forest decision-making.

The absence of the dynamic features from a traditional GIS is attributed to its built-in spatial model – the one that relies on a calculation of "vector" line intersections, and on an

extensive processing of topological relations. Analyses show that the "intersection finding" and "topology building" are hard to be operated in a local fashion. In contrast, the space-filling "Voronoi" tessellation had been shown (e.g. Gold 1991, 1994) to be able to handle local updates in a well-defined way. It was decided that this approach would be implemented, and then evaluated.

The research and implementation of the Voronoi tessellation produced a functioning prototype which showed the feasibility of the method. However, it became evident that the problems of storing the tessellation would emerge significantly for a full-scale operation in a seamless forest map. Attempts were made to address this issue as well. This led to the development of appropriate dynamic spatial partitioning methods, and the design of the associated hierarchical and dynamic GIS data model, which again was implemented at the prototype level.

On this basis, a preliminary design was developed for a forest data management system based on the dynamic data model, and the results were examined to see how closely the data model matched the forest harvest management needs as originally specified. Because of the generality of the method, its use in other applications was also briefly described.

1.5.2 The Development

Examine the need of the forest industry

The task is to understand the environment in which a forestry spatial database management and decision support system functions. For this purpose, the author searched the literature to find discussions about forestry, especially the value of forests to the society, the attitudes and positions of public, government, and forest companies towards forestry, and current applications of GIS to forestry. The author also talked with managers and professionals from forest companies on different occasions, and visited forest companies to observe GIS practices within the organizations. It is found that a sustainable forestry is to everyone's interest, but what is considered a sustainable forestry practice is very much debatable. There is no doubt, however, that the use of GIS can improve synthesising, managing, and

presenting information for informed decision-making and practice concerning forestry. The identification of characteristics of a sustainable forestry data management and decision support system is the result of the preliminary research, which helps with the specification of important features of the GIS for forestry. The research and discussions to justify the subject of the thesis are the content of this chapter.

Evaluate the weaknesses of traditional GIS

As a continuation of justifying the thesis research, questions about commonly practiced spatial data models in GIS are arisen. These concern the capabilities of spatial data models, and their strength and weakness in serving as the backbone of a GIS. In order to answer these questions, fundamental concepts involved in modelling geographical spaces and the mathematical background for spatial data modelling are reviewed. It becomes evident that there are usually two separate spatial data models involved in a GIS to support geometric and topological data management operations. Various spatial data models employed by most current GIS are studied, with respect to the requirements of an advanced GIS. The important shortcomings inherited in these conventional spatial data models are revealed from the research, which suggests that the GIS community has not found desirable solutions towards spatial data modelling. More investigations are needed in this field to meet contemporary challenges. The review of the theoretical background in this stage is reported in Chapter 2.

Suggest and implement a dynamic data model for GIS

The subsequent research is then concentrated on the dynamic Voronoi data model, which is introduced with its intuitive idea of modelling geographical spaces. The research is based on a well-known Voronoi data model developed by Dr. Gold in the Centre de recherche en geomatique at Université Laval. In a rigorous manner, the research delivers a complete account about the mathematical background, history, data structures, construction, temporality, and GIS applications of the dynamic Voronoi diagrams of points and line segments. This effort demonstrates the power of the Voronoi data model in solving geo-spatial problems frequently encountered in a GIS. It also prepared the author with the

necessary theoretical basis and experience to manipulate, question, and leverage the data model for advanced applications. This part is presented in Chapter 3.

Technical problems of the suggested model and solutions

The problems with the preliminary Voronoi diagram are primarily related to its storage structure. One of the tasks of the thesis research is to improve the storage structure such that spatially adjacent objects can be assigned similar addresses. If this can be achieved, the next task, dividing the entire storage structure into disk pages, would become possible

The method is based on observing patterns and behaviour of the data model. Since there is no precedent work on the storage issues with the Voronoi diagrams, no references can be resorted to help with ideas. At beginning, it was difficult to divide the Voronoi diagram into distinct chunks, because of the incremental construction. The arbitrary order of data input and spatial distribution could results a high degree of discrepancy between spatial patterns and the storage of the data. A few attempts were once considered, by resorting to some subordinate spatial data structures such as the R tree. They were soon abandoned because the Voronoi diagrams would be managed with conventional, rigid, and unintuitive geometric data structures. Besides, it would complicate the data model by imposing another spatial partitioning method, in addition to the one Voronoi uses.

The solution of the storage related problems relies on the deployment of the characteristics of Voronoi diagrams with line segments. In this step, the research is focused on the interactive behaviour of Voronoi diagrams, at both sides of a chain of line segments. It is noted that the change of the Voronoi diagram at one side of a chain does not affect the spatial configuration at the other side. Observations also show that interactions do happen in part of the storage structure on both sides, and that these interactions are possibly tractable. This finding encouraged the experiments of identifying objects first within rectangular polygons and then irregular polygons. After this, the research is directed to partitioning storage structures of Voronoi diagrams based on arbitrary polygons. The technique finally used to fulfil our tasks is reported in Chapter 4.

Propose a hierarchical and dynamic GIS data model

The next stage of the research defines spatial objects of the data model. The objects include the familiar ones (point, line, polygon), and the newly created ones, Voronoi objects (defined as Voronoi Map Objects or VMO in the thesis). Equally important in this stage are defining inter-relationships among spatial objects, the organization structure for them, and the operations upon these objects. The task in this stage is to set up the theoretical framework of the data model endeavoured in this thesis. The implementation of the VMO classes needs to be done by using an object-oriented method. Their geometric and topological properties and operations are realized through the Voronoi diagram. Chapter 5 covers the definitions and discussions about the data model.

Prototype a forest data management system

In the final stage, the data model will be used in the design of a prototype of a forestry data management system. The purpose is to demonstrate how the data model can be served under an object-oriented design and modelling environment. The capability of the data model will be re-examined with respect to the requirements specified in the first stage. The illustrative design of a forestry data management and decision support system is discussed in Chapter 6.

Before concluding in Chapter 8, we would like to note that the data model could also be used to develop other interesting applications, such as intelligent map generalization and parallel processing of spatial problems. This is extended in Chapter 7.

Chapter 2

Modelling Geographical Space

The "geographical space" is a phrase often referred to by professionals in GIS about which an agreed scientific definition is still missing. Nevertheless, sensing the existence of geographical space starts in childhood intuition and is enhanced by observation and education. The Earth, to our practical interests, is the totality of geographical space which host collections of concrete things such as territories seas or plateaus, mountains lakes and rivers, cities and towns; not-so-concrete concepts such as populations of people, animals, forests, distributions of ore materials, variations of weather, temperature; or illusive ideas which are related to things and concepts such as time, events; etc.. They are geographical entities which either physically occupy, are akin to, or are associated with parts of the Earth, and they all take part in geographical processes (natural or non-natural) some of which we have perceived as geographical phenomena. It is the desire to describe and understand geographical processes with the objective of exerting influence and control over them, based on natural laws, that motivates us to model geographical space.

2.1 Field- or Object-Based Views of Geographical Space

In modelling geographical space, it is necessary to consider artificial objects and simulated processes that we use to represent geographical entities and processes of interests. It is the question of what constitutes artificial objects as modelling spaces that has been the cause of the enduring debate between two fundamentally differing views [Chrisman 1975, 1978; Peuquet 1984; Herring 1991; Couclelis 1992]. So tremendous is the influence of the debate that our thinking, language of discourse, data models, and GIS systems have fallen into the main frames of the two different views.

The first view, labeled *field-based* or *position-based*, conceptualizes a geographical entity as a *set of locations* each member of which is associated with a *field* to represent a certain thing or phenomenon on the Earth. The set of locations is usually referenced with a *coordinated framework* which has an underlying geometric space. The field is described by one or more attributes each of which is mapped to some well defined attribute domain (e.g. a set of integers). The basic object in this view is therefore a location. A field is not a distinct object but a set of attribute values attached to the location set. Figure 2.1 illustrates such a model representing the distribution of trees in a forest stand 'F'. The location set is referenced by a grid network. The attribute modelled is the density of trees whose values vary from cell to cell.

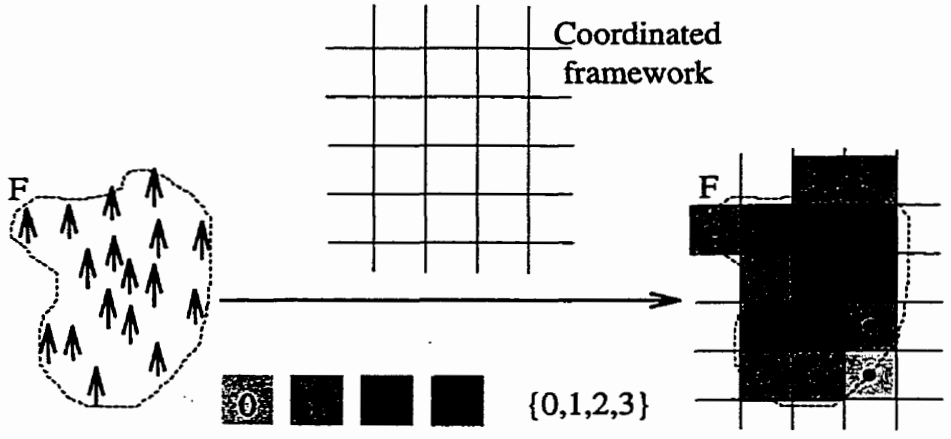


Figure 2.1 A field-based view of a geographical entity F

The second view, labelled *object-based* or *feature-based*, conceptualizes a geographical entity as a single identifiable *object* (or *geo-object*) associated with a few characterizing properties or attributes given as other objects. One of the properties of a geo-object describes its extension on the Earth, the result of which is referred to as its *referencing spatial object*. A spatial object has a structure embedded in a mathematical space. Therefore, a geo-object is *spatially referenced* if it is associated with a spatial object in a definable way. For example, a forest stand is a geo-object which is spatially referenced to a spatial object called 'polygon', in addition to other possible defining properties such as

'stand no.', 'plant time', and 'density rate' (Figure 2.2). Other examples of geo-objects and their referencing spatial objects can be listed: a highway, referenced by a 'line', a well by a 'point', etc. With this view, a geographical space is a collection of distinct geo-objects.

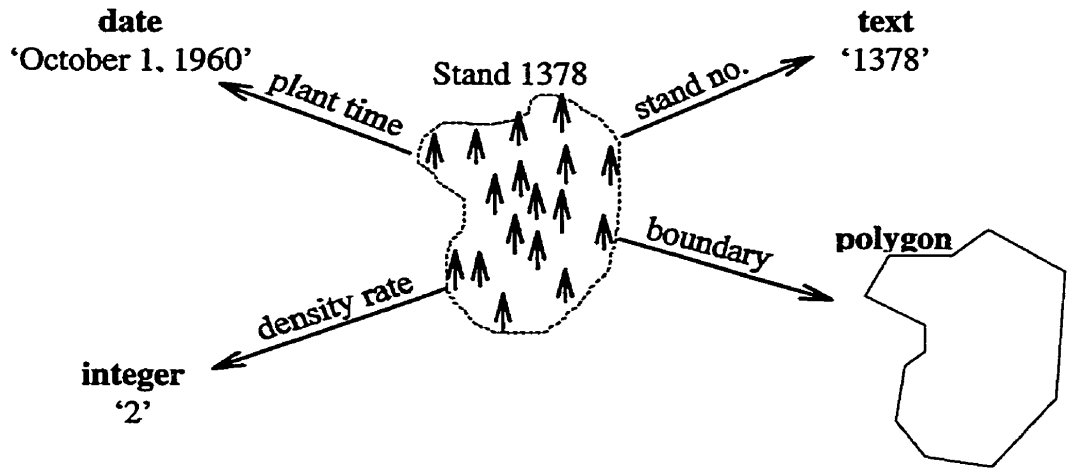


Figure 2.2 An object-based view of a forest stand (after Worboys [1995], pp. 165)

It is not the intention of this thesis to compare these two views. Generally, the field-based view is convenient to model geographical entities that do not have clear boundaries to "wrap up" their insides from their outsides. Instead, some "fuzzy" boundary patterns exist for a set of locations with respect to an attribute. As is shown in Figure 2.1, the attribute values indicate denser trees in central locations and sparser in the surrounding ones. The object-based view, on the other hand, relies on clear definitions of objects and is more appropriate if objects themselves are the major concern of a system. Actually, as Herring [1991] stated, the underlying need to distinguish pieces of geometry are logically equivalent. We are in favor of the concepts and terms exposed by the object-based view. The idea of defining a meaningful object with a small set of well defined more primitive objects is more compatible with the object-oriented modelling technology. It is possible, with the object-oriented approach, to bring both views of the real world into one representation. We will come back to this possibility later.

2.2 Spatial Objects

A special characteristic of geographical objects is that they are referenced to spatial objects which themselves must be specified. A *spatial object* is "spatial" because it exists inside a "space", called the *embedding space*. The specification of a spatial object depends upon the structure of its embedding space. The most commonly used embedding space in geographical applications is *Euclidean space*. Using the terminology of set theory, a Euclidean space is a product set of real numbers. The *dimension* of a Euclidean space is counted by the number of sets of real numbers R participating in the product. We restrict our discussion in this thesis to a two-dimensional Euclidean space (or the *Euclidean plane*), denoted R^2, which is therefore a collection of ordered pairs of real numbers. We shall also assume that in the Euclidean plane we have a *Cartesian coordinate system* for which there is a pair of orthogonal reference axes. The intersection of the axis pair is the *origin* of the coordinate system and the unitary segments on respective axes have identical length. With this, we define the following primitive spatial objects:

Point: A point a is defined by a unique pair in R^2, denoted $a(x_1, x_2)$ with $x_1, x_2 \in R$.

Line: Given two distinct points a and b in R^2, a line l incident with a and b, denoted $l(a, b)$, is defined as the pointset, T. That is

$$T_\lambda = \{\lambda a + (1-\lambda)b \mid \lambda \in R\}.$$

Line segment: If we constrain the range of parameter λ to be in [0, 1] for the pointset T, we get a line segment which is a closed pointset

$$T_{[0,1]} = \{\lambda a + (1-\lambda)b \mid \lambda \in R, \lambda \in [0, 1]\}$$

The points a and b are called the *endpoints* of the line segment.

Polyline: A polyline in R^2 is defined as a finite set of line segments such that each endpoint is incident to exactly two line segments, except possibly for two endpoints, called the *extremes* of the polyline. A polyline is *simple* if no two line segments intersect at any place other than at their endpoints. A polyline is *closed* if it has no extremes.

Polygon: A (*simple*) polygon in R^2 is defined as the area enclosed by a simple closed polyline. The polyline forms the *boundary* of the polygon. Each endpoint of a line segment of the polyline is called a *vertex* of the polygon. The area enclosed by the polygon boundary is the *interior* of the polygon. In common sense, the term polygon is frequently used to denote the union of the boundary and of the interior. The polygon in this sense is *bounded*. Naturally, the rest of the plane, that is, the complement of the bounded polygon, is the area *exterior* to the boundary and is *unbounded*. Therefore, a simple boundary partitions the plane into two distinct regions, the interior and the exterior of the polygon (the *Jordan curve theorem* [e.g. Christenson and Voxman 1977]).

2.3 Properties of Spatial Objects

Just like any other artificial object used to model a system, spatial objects possess properties which: 1) have some measurements on spatial objects (metric properties); 2) classify similar groups of spatial objects (geometric properties); and 3) describe relationships between spatial objects (topological properties). In this section, we discuss those properties consistent with the Euclidean plane.

Metric properties. Metric properties are dependent on the metric distance, to be defined below, between two points in R^2:

The *distance function* $d : R^2 \times R^2 \rightarrow [0, \infty) \subset R$ is called a *metric*, if the following conditions hold for all points $x, y, z \in R^2$:

1) $d(x, y) = 0 \Leftrightarrow x = y$ (reflexive)

2) $d(x, y) = d(y, x)$ (symmetric)

3) $d(x, y) \leq d(x, z) + d(z, y)$ (triangle inequality)

For example, the distance function d defined by

$$d(x, y) = \sqrt{[(x_1 - y_1)^2 + (x_2 - y_2)^2]},$$

for $x(x_1, x_2)$ and $y(y_1, y_2)$, is a metric and is called the *Euclidean distance*.

Examples of the metric properties of spatial objects include: the Euclidean distance between any two points on a polyline; the perimeter and the area of a polygon; the centroid of a polygon or a cluster of points; the angle between two lines; and the bearing of a line in respect to one axis. These properties can be used to measure other properties of associated geo-objects such as: the average flow speed of a river between two paper miles by the river; the average log volume of a forest stand; etc. Another interesting use of metric properties is illustrated in Figure 2.3 where clusters of points in a point set S can be classified as identifiable objects given a distance δ (δ-clusters). For each cluster, the δ-radius circle, centered at each point, has an intersection with some other δ-radius circles. The distance function d can also be used to define a topology for R^2. This topology is called the *metric topology* or the topology induced by the metric d. We will soon see what we mean by "topology".

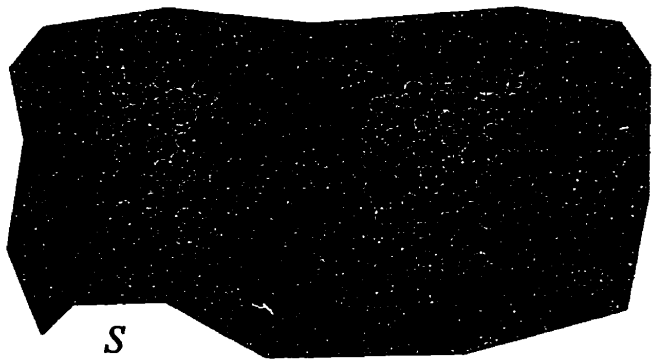


Figure 2.3 Clustering a point set given a distance δ

Geometric properties. While metric properties are quantitative, i.e. they can be expressed by values, geometric properties are qualitative, i.e. they are instead expressed by defined characteristics regarding their forms. Identifying geometric properties helps to classify spatial objects into groups with similar geometric structures, thus facilitating the design and development of geometric algorithms to calculate the metric properties of spatial objects or to solve geometry problems. The study of special geometric properties and the methods to identify them is the subject of *computational geometry* [Preparata and Shamos 1985]. The most often used geometric properties of spatial objects are based on the concepts of visibility, convexity, and monotony which are defined below:

Visibility: Let S be a set of points in R^2. Then a point x in S is *visible* from point y in S if either $x = y$ or it is possible to draw a straight line segment between x and y that consists entirely of points in S. If the point x in S is visible from every point of S, the point x is called an *observation point* for S.

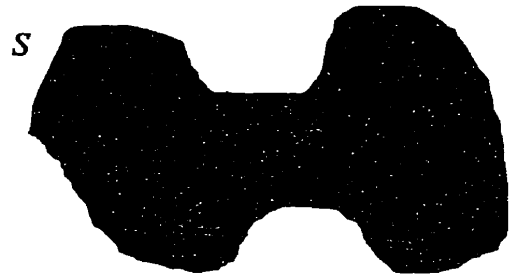


Figure 2.4 Visibility between points x, y, and z (after Worboys [1995], pp. 110)

Denote visibility as a relation V on R^2. The relation V is reflexive, symmetric, but not transitive. Figure 2.4 demonstrates an example of the visibility relationship V between x, y, and $z \in S$ in R^2. The example shows invisibility between points y and z.

Convexity: Let S be a set of points in R^2. The set S is *convex* if every point of S is an observation point for S. It is *semi-convex* (*star-shaped* if S forms a simple polygon) if there is some observation point for S. The collection of all observation points for S forms the

kernel of S. It is a natural deduction that the kernel of S must have a convex shape since all observation points for S are also visible to each other.

The study of the convexity of point sets in R^2 is of great value in computational geometry. For a non-convex or semi-convex point set S in R^2, the overall convexity of the set may be gained or enhanced by decomposing S into a finite number of convex or semi-convex subsets. Alternately, larger convex regions containing S may be obtained, of which the boundary of the smallest convex region containing S is called the *convex hull* of the point set S in R^2. Figure 2.5 illustrates a convex polygon (a), a semi-convex polygon (b), a not-semi-convex polygon (c), and a convex hull aggregating a collection of disjoint polygons (d).

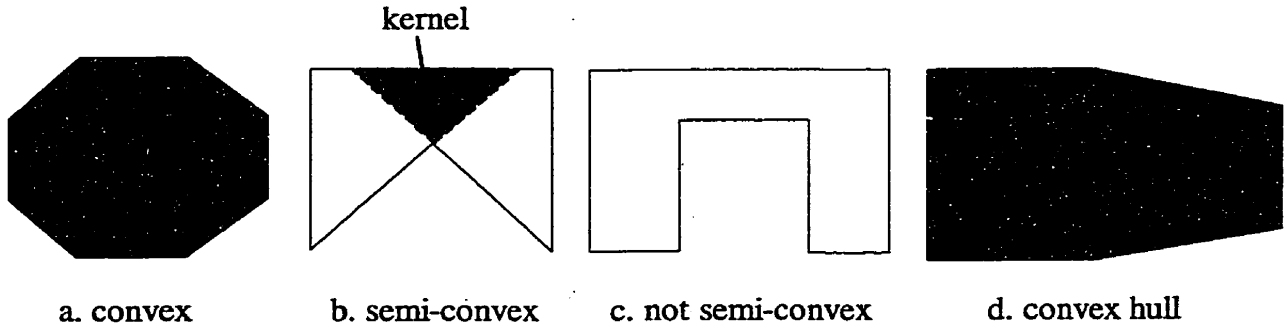
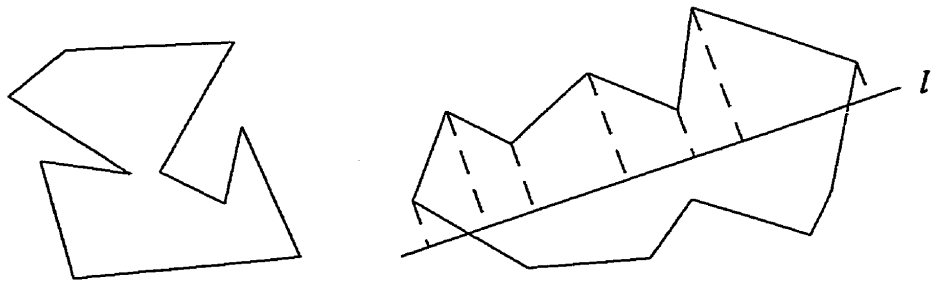


Figure 2.5 Convexity and convex hulls

Monotony: The monotony of a polygon is dependent on the definition of a monotone chain. Let chain $C = (p_1, p_2, ..., p_n)$ be an ordered list of n points in R^2. The C is said to be *monotone* if and only if there is some line in R^2 such that the projection of the vertices onto the line preserves the ordering of the list. A polygon is a *monotone polygon* if its boundary may be split into two polylines, such that the chain of vertices of each polyline is a monotone chain. Figure 2.6 shows two polygons, one is not monotone (a) and the other is monotone (b).



a. not monotone b. monotone with respect to line l

Figure 2.6 Monotony of polygons

Topological properties: These properties describe a set of particular relationships between one object and another embedded in the Euclidean plane R^2. They are particular because they remain the same by allowing certain operations acted upon the embedding plane to stretch, contract, and twist, but not to tear and fold it. Imagine the Euclidean plane to be an unbounded sheet of rubber. Also imagine spatial objects to be figures drawn on this rubber sheet, particularly one point drawn inside a polygon. The fact that the point is "inside" the polygon will remain unchanged no matter how the rubber sheet is stretched. The "insideness" is therefore a topological property. Though not explicitly examined, there are some other relationships in the example which are qualified to be topological. For instance, any vertex of the polygon remains "on" the boundary. Furthermore, the "incidence" relationship between a vertex and a line segment is also preserved. The operations involved in stretching or contracting the embedding plane are called *topological transformation* or *homeomorphism.* With this understanding, a *topological property* can be defined as one that is preserved by topological transformations of the space. The study of topological transformations and the properties that are left unchanged by them is the subject of *topology.* Particularly, the topology that is based on the rubber sheet analog is referred to as *usual topology* for the Euclidean plane.

In contrasting with topological properties, the metric and geometric properties mentioned previously are not invariant under topological transformations based on the usual topology. Metric properties are invariant only under the translation operation. The size of an area, for example, will be changed when the embedding space is stretched wider. Geometric

properties can withstand rigid transformations such as translation, rotation, and scaling, but are sensitive to uneven stretching or contracting of the embedding space. This can easily be verified by dragging one part of the lower concave ridge of Figure 2.4. Pulling down, the rubber sheet (embedding plane) would become flatter and points y, and z would be visible (Figure 2.7).

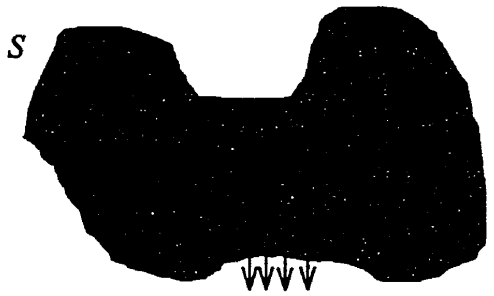


Figure 2.7 Visibility changed by dragging and pulling the rubber sheet

Familiar topological properties between spatial objects in R^2 generally considered in developing spatial information systems include *neighbourhood*, *connectivity*, *containment*, and *adjacency*. Examples translated similarly from reality corresponding to these properties might be listed and illustrated (Figure 2.8) below:

Neighbourhood -- What are the parcels bordering lake A?

Connectivity -- Is it possible to travel from A to B by car?

Containment -- What islands are on lake A?

Adjacency -- Which highways branch out from city C?

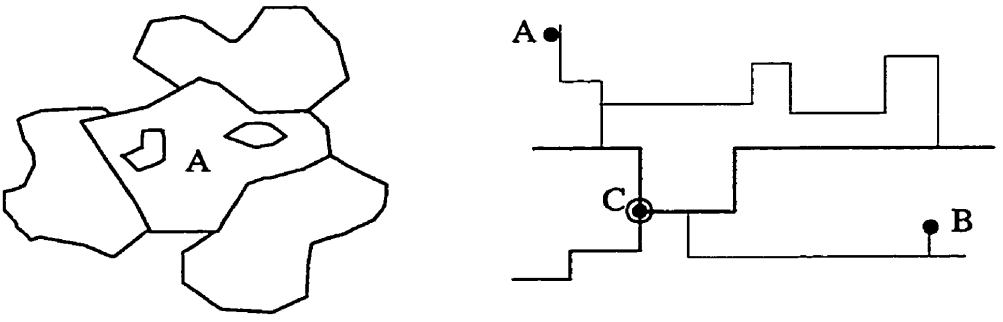


Figure 2.8 Some topological properties

The study of topological properties and transformations is divided into two branches known as *pointset* (or *analytic*) topology and *algebraic* (or *combinatorial*) topology. The rudimentary objects and tools for pointset topology are point sets and neighbourhoods, respectively. A new set is constructed by defining a neighbourhood relation on a given basic set. A set X, together with a collection of definable neighbourhood U, denoted (X, U), becomes a *topological space*. One of the important aspects of pointset topology studies homeomorphical transformations between topological spaces, typically homeomorphisms between a new topological space and one of a handful of well defined and well studied topological spaces. *Topologically equivalent* spaces should have equivalent topological properties. Thus the topological properties of a new space can be relatively easily revealed. Fundamental notions and facts of pointset topology include *connected*, *open*, *closed*, *compact*, and *continuous*.

Although the basic properties of topological spaces and concepts developed in pointset topology may seem not to correspond directly to the obvious topological ideas about spatial objects, understanding them is critical to capture the essence of topology and they are helpful to construct sound, faithful, and lasting spatial data models. In geographical applications, any sets of spatial objects must embed in some topological space. The insightful statement put by Worboys [1995] (pp. 118) is worth being kept in mind: "it is not possible to consider the topological properties of sets in exclusion from the larger spaces in which they are embedded." Important notions of topology are applied to describe our data model proposed later in this thesis. We will discuss more on this later.

The algebraic topology takes a more tractable approach by emphasizing the combinatorial nature of finite "chunks" of spaces and functions that relate them. This is the basic approach taken by Euler to solve the famous Konigsberg Bridge Problem [e.g. Bollobås 1979]. It turns out that for figures built from such chunks (simplexes) of dimension ≤ 3, the combinatorial relationships reflect all relationships which are topologically possible [Stillwell 1993]. In the theory of the simplicial complex, the most primitive building blocks for constructing figures are the simplest elements called *n-simplex* existing for any

dimension n. Thus an 0-simplex exist for any point, an 1-simplex for a closed line segment, a 2-simplex for a triangle, a 3-simplex for a tetrahedron, etc. Any n-simplex is composed of $(n+1)$ *geometrically independent* simplexes of dimension $(n-1)$. For example, a triangle, a 2-simplex, is bounded by three 1-simplexes. These 1-simplexes are geometrically independent if no two triangle edges are parallel and no edge is of length 0 [Giblin 1977]. Each subset of $m+1$ points from a n-simplex similarly determines a m-simplex which is called a *face* of the n-simplex. For example, the faces of a 2-simplex include three 1-simplexes and three 0-simplexes which are edges and vertices of the triangle, respectively. The union of the $(n-1)$-dimensional faces is called the boundary of the n-simplex, so all lower-dimensional faces lie in the boundary. The boundary of a triangle is the union of its three edges. As an exercise, we illustrate an operation on complexes following the style presented in Worboys [1995] (pp. 128-131).

With the building blocks, any larger structures can be constructed by pasting together simplexes so that faces of a given dimension are either disjoint or coincide completely. This forms a simplicial complex. A n-dimensional *simplicial complex* (or simply *n-complex*) C is a finite set of simplexes of dimension $\leq n$ satisfying the following conditions:

a) A face of a simplex in C is also in C; and
b) The intersection of two simplexes in C is either empty or is also in C.

Figure 2.9 shows two 2-complexes A and B, whose simplex sets are

$$A = \{a,b,c,d,e,f,ab,bc,cf,fd,ac,bd,ec,fc,dc,abc,bec,efc,fdc,dac\}$$
$$B = \{g,h,i,gh,hi,gi,gih\}$$

respectively. Note that the complex B coincides with its 2-simplex constituent *gih*.

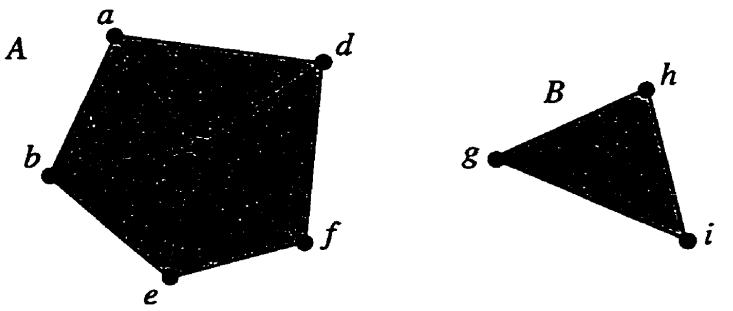
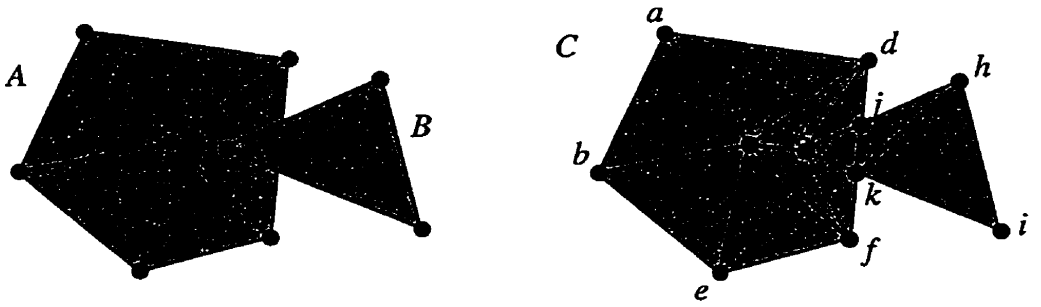


Figure 2.9 Two 2-complexes

Now we intersect the two 2-complexes A and B (Figure 2.10a). The result should also satisfy the two conditions above. To this end, two 1-simplexes need to be introduced at the intersections (points j and k in Figure 2.10b), which future induces 1-simplexes and 2-simplexes. The resulting 2-complex C is the simplex set satisfying both conditions.

$$C = \{a,b,c,d,e,f,g,h,i,j,k,ab,be,ef,fk,ki,ih,hj,jd,da,ac,bc,ec,fc,dc,da,dg,gf,gc,gk,gj,jk,$$
$$abc,bec,efc,fgc,fkg,gdc,gjd,gkj,dac,kih,khj\}$$



a. Intersecting A and B

b. The resulting complex C

Figure 2.10 Intersecting two 2-complexes to obtain a new complex

The power of the combinatorial approach lies in the algebraic interpretation of the boundary operation for a complex C. The boundary of a n-complex C, ∂C, is a $(n-1)$-complex. To calculate the boundary of a complex, the orientation of n-simplices (for $0 < n < 3$) is introduced (Figure 2.11). The orientation of an 1-simplex ab can be directed either

from a to b or from b to a. The orientation of a 2-simplex determines that of all its constituent 1-simplexes. An *oriented complex* is one in which every simplex is consistently oriented.

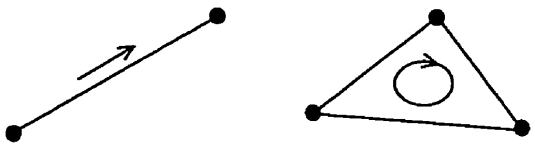


Figure 2.11 The orientations of simplexes

With orientation, it is possible to define the boundary operation in a pure algebraic way. Let the 0-simplex x, be denoted x; the oriented 1-simplex, starting from x and ending at y, be xy, and $-xy$ be yx if it is oriented the other way; and the oriented 2-simplex, with the vertices in ordering being x, y, and z, be xyz. Introduce any linear combination of simplexes (called a *chain*) of the form $a_1S_1 + ..., + a_mS_m$ where the S_1, ..., S_m are simplexes and the a_1..., a_m are integers. The boundary operator ∂ on the chain is defined as

$$\partial(a_1S_1 + ..., + a_mS_m) = a_1\partial(S_1) + ..., + a_m\partial(S_m)$$

The boundary operator for a n-simplex (for $0 < n < 3$) is as the following:

$$\partial(x) = 0$$
$$\partial(xy) = y - x$$
$$\partial(xyz) = xy + yz + zx$$

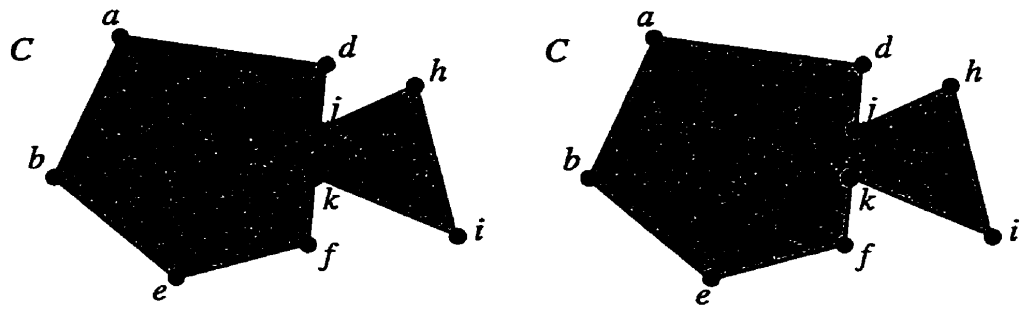
Apply these rules to the complex C in Figure 2.10. We first assign a consistent orientation to each of the 2-simplexes in C which results in the diagram in Figure 2.12a. Next we express the chain of the complex to be a linear combination of its 2-simplex constituents, that is

$$C = bac + bce + ecf + fcg + fgk + gcd + gdj + gjk + dca + kjh + khi$$

Now applying the boundary operation we get

$$\partial(C) = ba + ad + dj + jh + hi + ik + kf + fe + eb$$

This results in the boundary of the complex C in Figure 2.12b. One may verify that the 1-complex boundary chain forms a simply closed polygon ordered in clockwise sense.



a. Orienting each 2-simplex in C b. The boundary of the complex C

Figure 2.12 The boundary of a 2-complex C

As one may have observed, the theory of complexes actually decomposes any object in the Euclidean plane into a set of 2-simplexes, that is, a triangulation. This approach is assured by a theorem first proved by Radò in 1925 which asserts that any bounded Euclidean space is triangulatable. The triangulation observing the two conditions mentioned previously are not unique for a given figure. In Chapter 3, we will introduce another way of triangulating the Euclidean plane which produces a unique triangulation.

2.4 Representations of Spatial Objects and Relationships

Representing spatial objects and relationships in a tractable structure is a critical step to constructing a computerized database for geographical applications. There are two aspects to consider, usually separate. One concentrates on the topological relationships expressed

by identifiers (system generated IDs) of spatial objects. The other specifies detailed geometric embedding of spatial objects in terms of identifiers as well as their coordinates. The reasons for this dichotomy are:

Firstly, topological relationships between spatial objects, once identified and coded, are not strongly bound to their geometric embedding. Point A is in polygon P no matter where A is positioned in the interior of P. Likewise, the adjacency of two polygons does not care how these two polygons are configured (vertices and shapes).

Secondly, topological operations can be expressed entirely at the identifier level without involving coordinates. An example is the boundary operation presented in the previous section. On the other hand, metric and geometric operations rely on the detailed embedding of objects.

Thirdly, due to the state of the art of computerized spatial data handling, some topological representations of spatial objects are readily compatible to current general database schema and structures where other aspatial data usually reside. This is in contrast with the geometric structures of spatial objects which are proved inefficient when general database structures are employed to store them.

This section discusses commonly practiced topological structures which explicitly capture some topological relationships. Geometric structures concerning the embedding of spatial objects and their storage will be discussed in the next section.

It turns out that most of the current topological structures for a Euclidean plane are based on the *planar graph*. As shown in Figure 2.13, the primitives in a general planar graph are nodes and arcs. An arc consists of arbitrary many line segments embedded in the plane. Arcs can only intersect at nodes of the graph. If an arc intersects itself at one node, it forms a loop. Multiple arcs must share two identical nodes. An arc dangles as one of its nodes is not adjacent to any other arcs. If an arc has two dangling nodes, it is floating. A closed chain of arcs forms a polygon. A polygon can contain islands. Two arcs are said to be

adjacent to one another if they are both incident to the same node. Two nodes are said to be adjacent if there is an arc joining them. A path is defined as a sequence of arcs which can be followed continuously without any arc being used more than once. A path is closed when it starts and finishes at the same node; otherwise it is open. A closed path is also called a circuit. The order of a node is the number of arcs incident to it. If an arc meeting a node is a loop, then that arc is counted twice in calculating the order of the node. A connected graph is a graph for which every pair of nodes can be joined by a path. If a graph is not connected, then it consists of several pieces called its components.

The relationship used to constraint elements of a planar graph with λ components is expressed by the *Euler formula*. That is

$$\sum_{i=1}^{\lambda} (p_i - a_i + n_i) = 2\lambda$$

where i represents the ith component of the graph ($1 \leq i \leq \lambda$), while p_i, a_i, and n_i represent the numbers of polygons, arcs, and nodes in each component, respectively. Note that in this formula, each complimentary region of a component is counted as one polygon. As an example, the planar graph in Figure 2.13 contains 3 components (the floating arc, the island, and the rest of the graph). The total number of polygons is 10 with the compliant area being counted three times; the total number of arcs is 15; and the total number of nodes is 11. It is easy to verify that the Euler formula holds for this graph.

One of the earliest topological representations which found practical application in the 1970's and is still influential today is the TIGER structure, meaning Topologically Integrated Geographic Encoding and Referencing. This structure was developed by the US Bureau of the Census and is described by Boudriault [1987]. The TIGER structure is actually the refined successor of older chain-node structures, such as DIME [US Bureau of the Census 1970], also developed by the US Bureau of the Census in 1967 and POLYVRT [Peucker and Chrisman 1975], developed at the Harvard Laboratory for Computer Graphics

and Spatial Analysis. The now well-known Node-Arc-Polygon representation is the DIME-like structure.

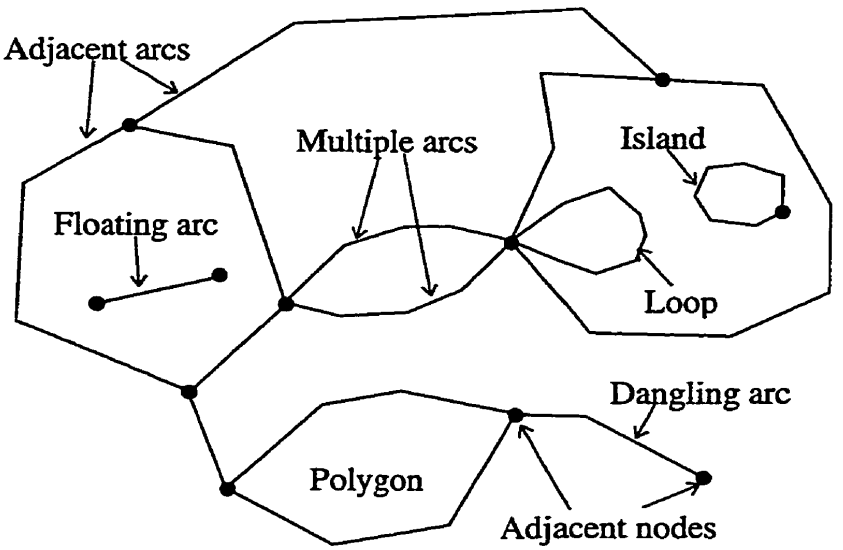


Figure 2.13 A general planar graph

Node-Arc-Polygon (*NAP*): The NAP structure represents explicitly the adjacency relationships between polygons. It can be conveniently modelled by the ER diagram and represented with one relational table. The constituent entities are the directed arc, node, and polygon which are constrained by the following rules:

- Each directed arc has exactly one start and one end node.
- Each node must be the start node or end node (maybe both) of at least one directed arc.
- Each polygon is bounded by one or more directed arcs.
- Directed arcs may intersects only at nodes.
- Each directed arc has exactly one polygon on its right and one polygon on its left.
- Each polygon must be the left or right polygon (maybe both) of at least one directed arc.

An EER diagram of the NAP structure is shown in Figure 2.14. It is possible to represent loops, islands, and dangling arcs with the above rules. However, single points with no arcs attached are not allowed with this representation. An example of the planar graph with

directed arcs is illustrated in Figure 2.15. The relational table for the configuration is given in Table 2.1. The EER diagram of the NAP can be extended to include details of the embedding. An example of the EER diagram with the embedding is given in Worboys [1995] (pp. 195).

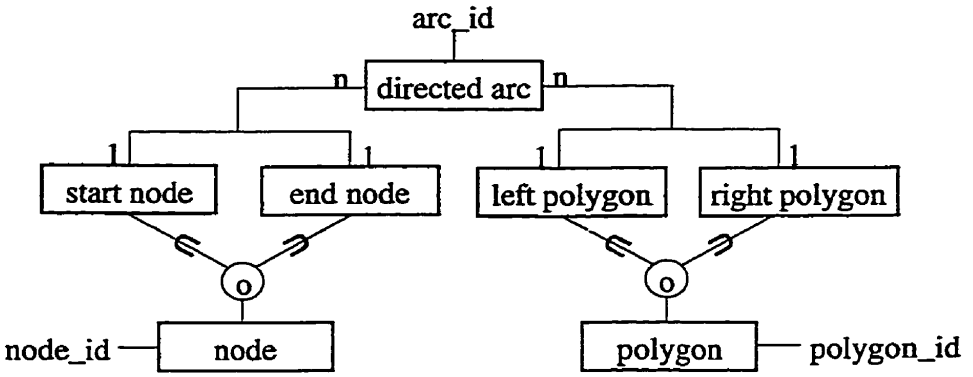


Figure 2.14 An EER diagram of the NAP structure

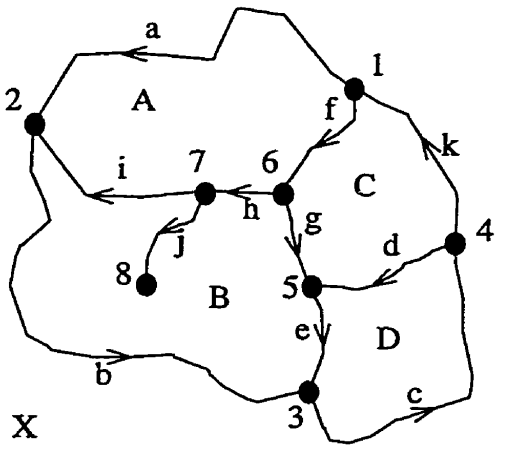


Figure 2.15 A planar graph with directed arcs (after Worboys [1995])

Arc Id	Start node	End node	Left polygon	Right polygon
a	1	2	A	X
b	2	3	B	X
c	3	4	D	X
d	4	5	D	C
e	5	3	D	B
f	1	6	C	A
g	6	5	C	B
h	6	7	B	A
i	7	2	B	A
j	7	8	B	B
k	4	1	C	X

Table 2.1 The NAP relation for the graph in Figure 2.15 (after Worboys [1995])

Doubly-connected-edge-list (DCEL): The DCEL [Muller and Preparata 1978] is similar to the NAP in preserving polygon adjacency. In addition, the DCEL adds two pieces of information to each directed arc: the previous and the next arcs (Figure 2.16). The previous

arc is associated with the start node and is the first arc encountered by cycling around the start node in an anticlockwise direction. The next arc is associated with the end node and is the first arc encountered by cycling around the end node in an anticlockwise direction. Figure 2.17 is the modified EER diagram from Figure 2.14 with two additional relations added for the DCEL to encode the previous arc and next arc information. Table 2.2 shows the DCEL in a relational form representing the configuration in Figure 2.15. The additional information facilitates searches for the arcs incident to a given node or the arcs enclosing a given polygon. However, special care must be taken when dangling arcs exist. As is shown in Figure 2.15, the directed arc *j* is a dangling arc. This fact can be identified from the tuple of the relation corresponding to arc *j* in Table 2.2 as the next arc is itself.

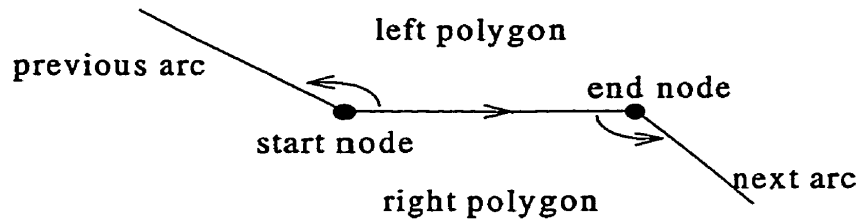


Figure 2.16 Relationships to a single arc in the DCEL

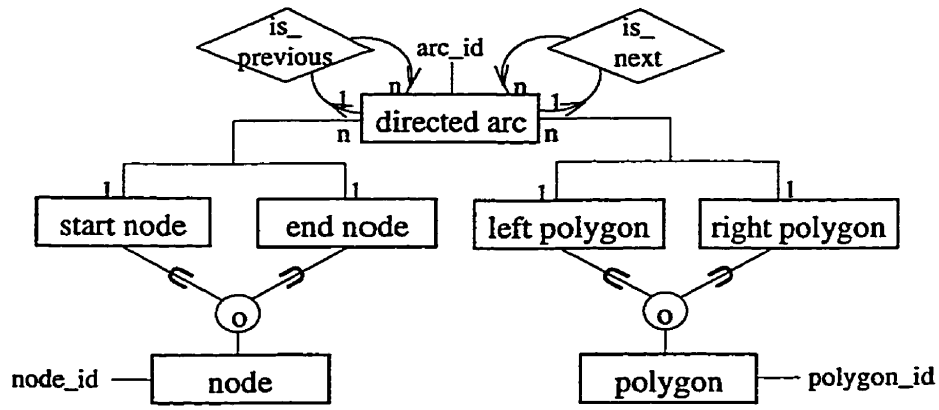


Figure 2.17 The EER diagram of the DCEL representation

Arc Id	Start node	End node	Left polygon	Right polygon	Previous arc	Next arc
a	1	2	A	X	f	b
b	2	3	B	X	i	c
c	3	4	D	X	e	k
d	4	5	D	C	c	g
e	5	3	D	B	d	b
f	1	6	C	A	k	h
g	6	5	C	B	f	e
h	6	7	B	A	g	i
i	7	2	B	A	j	a
j	7	8	B	B	h	j
k	4	1	C	X	d	a

Table 2.2 The DCEL relation for the graph in Figure 2.15

Object-DCEL: The object-DCEL [Stell and Worboys 1994] is aimed at encoding the "weak" connectedness for an aggregation of spatial objects each of which is a single "strongly" connected areal object. Two areal objects are *weakly connected* if they can be disconnected by removing a finite number of points. Otherwise, they are *strongly connected*. The following rule is observed in constructing the object-DCEL: the direction of the arcs is made so that the object's area is always on the right of each arc. Because of this rule, the relational representation of the object-DCEL is simplified. No polygon identifiers are needed since it is assumed that the whole structure represents a single areal object. Also, end node information can be omitted because the next arc starts from the end node of the previous arc. Figure 2.18 shows a weakly connected areal object structured with the object DCEL rule. The corresponding relation table is given in Table 2.2. The boundary of each strongly connected areal component can be easily retrieved from the table.

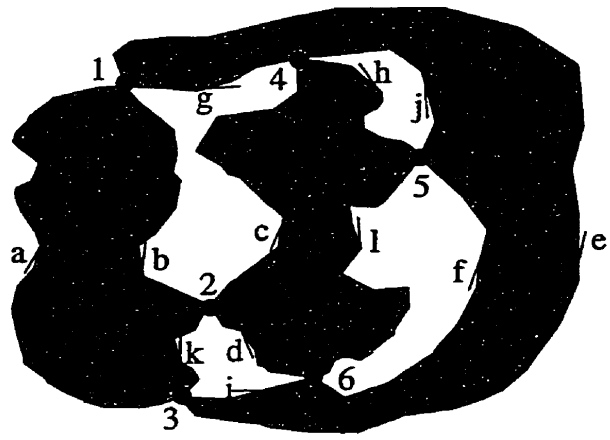


Figure 2.18 A weakly connected areal object

Arc	Start_node	Next_arc
a	3	b
b	1	k
c	2	h
d	6	c
e	1	i
f	6	j
g	4	e
h	4	l
i	3	f
j	5	g
k	2	a
l	5	d

Table 2.3 The Object-DCEL relation for Figure 2.18

2.4 Geometric Structures of Spatial Objects

The conceptual representations of spatial objects are capable of providing a concise description of the components of a spatial object and some topological relationships between spatial objects without caring too much about the geometric embedding. Achieving such a capability, however, depends on the coordinates that actually "fix" spatial objects in the space. Encoding the fact that two arcs are adjacent in a topological structure, for instance, requires a geometric judgment which recognizes that these two arcs intersect at an identical point. Therefore, geometric embedding of spatial objects must be present before the contents of a topological structure can be filled within a Euclidean space. It is obvious that capturing the geometric and metric properties of spatial objects requires their coordinates. As is pointed out by Herring [1991], the ability to represent the geometry of entities, or to distinguish the extent of the validity of an attribute value is a universal requirements of all GIS systems. Therefore, any spatial data handling system, whether having an explicit topological representation of the data or not, must be equipped with a mechanism of storing geometric embedding of spatial objects.

Geometric structures of spatial objects (or *geometric data structures*) are schemes that determine how *spatial data* (identifiers and coordinates of spatial objects) should be stored in a computer. A scheme of a geometric data structure may or may not actually consist of spatial data but emphasizes formation of methods to dispatch spatial data to physical storage in computers. For this reason, a geometric data structure is often referred to as a *spatial indexing structure*. One of the main objectives of devising such a scheme is that particular pieces of information about object embedding can be readily accessible without error or too much delay. Usually, access requirements come with spatial queries which can be topological and geometrical. Geometrical queries ask questions whose answers must rely on the spatial references of objects. Typical types of geometric queries seen in GIS include exact match, partial match, orthogonal range, and polygon queries [Mehlhorn 1984] which can be generalized into two types: point query and range query. A *point query* asks for one or more object whose spatial references are located at or proximate to a given point. A *range query* inquires for all objects whose spatial references are located within a given range of any shape.

Concerns about the design of a geometric data structure may be addressed via the following requirements:

1. Since geographical applications generally involve large quantities of spatial data, the geometric data structure needs to facilitate efficient retrieval of interesting spatial objects without a big ratio of overheads. Here the overhead means the number of uninteresting objects searched for in a query.
2. The relationship between the storage structure of topological representations and that of the geometric embedding should have a high degree of integrity.
3. The geometric data structure should not require some arbitrary or unnatural fragmentation of complex spatial objects.
4. The geometric data structure needs to be dynamic to update the embedding of spatial objects, and any modification causing the change of topological status should be captured and reflected promptly in the topological representation without halting the running of applications.

5. The geometric data structure should be partitionable to disk pages such that part of the structure can be loaded into memory and worked with independently. A *disk page* is the amount of data that can be continuously read, usually in one movement of the read/write head of the disk assembly. One disk page may be composed of one or more disk blocks (usually adjacent but this may be disrupted in a dynamic situation).
6. The ideal geometric data structure should have the ability to represent spatial data at different levels of detail to meet the needs of cartographic generalization.

Over nearly three decades, a variety of spatial indexing structures have been developed (see [Knuth 1973], [Günther 1988], [Samet 1990], and [Worboys 1995] for a survey). They can be broadly categorized into three families: linear orderings, bucket structures, and trees. In what follows, these three families of geometric data structures will be briefly examined. Typical examples from each category are used to illustrate their common characteristics. The purpose of doing this is to get an idea of how these geometric data structures are constructed and whether they encompass the above concerns.

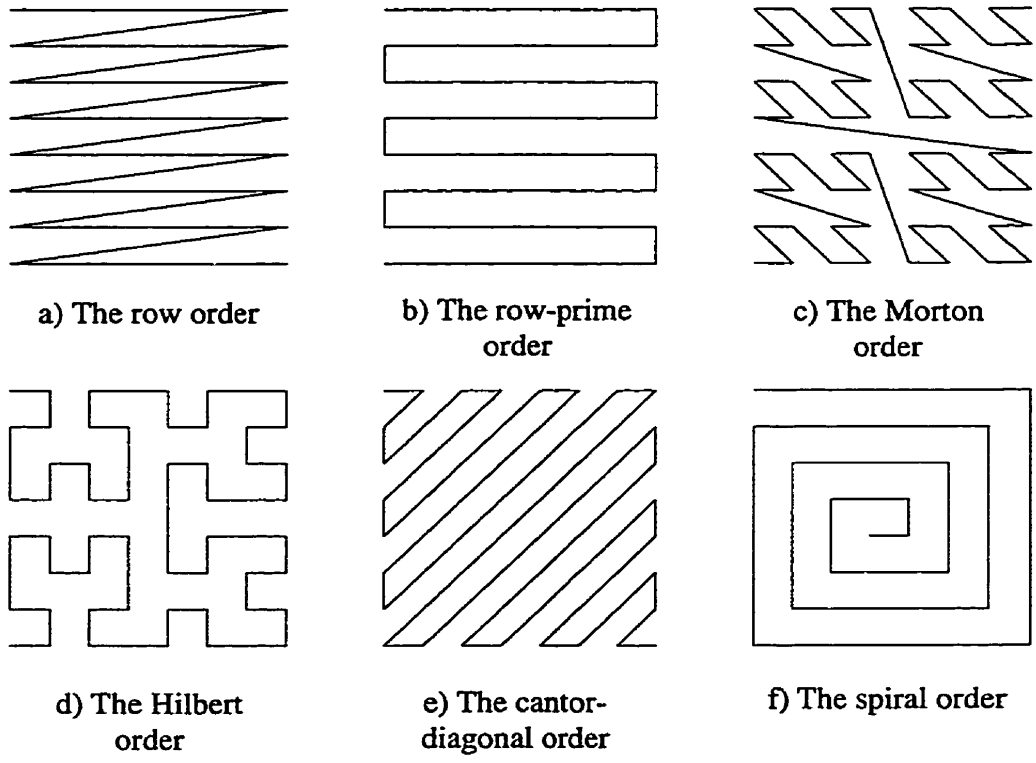


Figure 2.19 The linear orders

Linear orderings are the simplest geometric data structures in comparison to the other two families. They originate from scanning an analogous image and storing the texture information of pixels as a linear list. Each record of the list contains a location code which is the address of a pixel. This technique is extended typically to order k-dimensional vectorized points, thus transforming a k-dimensional problem into one-dimension. Well-known data structures for one-dimensional storage and retrieve such as the B-tree, the B^+-tree, ISAM, and hashing files can then be used. Popular orders include the *row order*, the *row-prime order*, the *Morton order* [Morton 1966], the *Hilbert order* [Goodchild and Grandfield 1983], the *spiral order* and the *Cantor-diagonal order* [Mark and Goodchild 1986] (Figure 2.19). The Morton order is also called the *Z order* [Orenstein 1983] or *N order* [White 1983], depending on how a location code is interleaved with the coordinates of a point. Comparisons between orders have been carried out based on the following evaluations: efficiency for spatial searches [Abel and Mark 1990; Yang 1992]; degree of spatial auto-correlation [Goodchild and Grandfield 1983]; number of consecutive location codes falling in a rectangle window [Jagadish 1990]; and practical considerations [Samet 1990].

Linear orderings are especially convenient for devising efficient range search algorithms for two or higher dimensions. Examples of range search algorithms based on the Morton and other orderings have been reported in Orenstein [1983], Yang [1992], and Stefanakis [1994]. Besides storing discrete points, the ordering techniques can also be applied to index cells containing complex objects.

Bucket structures flatly partition the Euclidean plane into rectangular cells, each having a location code as its address. Spatial objects falling in one cell are stored in a contiguous area of secondary storage. The simplest partition scheme produces a grid of fixed size for all the cells. In this case, linear orderings can be applied to transform the location codes of cells into one-dimensional lists. The major disadvantage of a fixed grid partition is that some cells may contain no data. This problem becomes more acute if the distribution of spatial objects is less uniform. The general approach to overcome this shortcoming is to use

some index compressing techniques to reduce the number of location codes. This method essentially transforms the fixed grid partition scheme into a more flexible one. A classical scheme following this approach is the Grid file [Nievergelt et al. 1984].

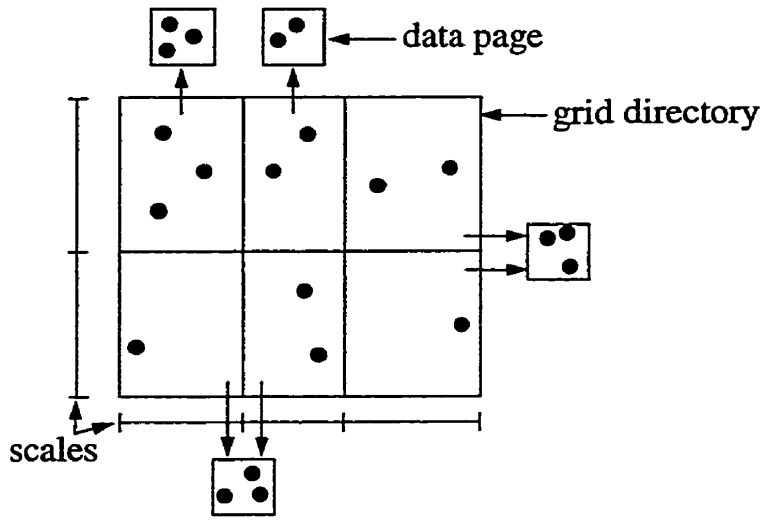


Figure 2.20 The Grid file structure

The Grid file (Figure 2.20) uses a grid directory consisting of grid blocks. All records in one grid block are stored in the same bucket (a disk page). Several grid blocks can share a bucket as long as the union of these grid blocks forms a k-dimensional rectangle. The grid directory consists of two parts. The first is a dynamic k-dimensional array, which contains one entry for each grid block. The values of the elements are pointers to the relevant data buckets. The second part of the grid directory is a set of k-dimensional arrays called linear scales. These scales define a partition of the domain of each attribute and enable the accessing of the appropriate grid blocks. The Grid file is designed to expand or contract as new data are inserted and deleted. A rectangle may be divided if it becomes too full and may be amalgamated with neighbouring ones if the space becomes too empty.

Tree structures are based on recursive decomposition of space into a hierarchical indexing of cells. Depending on particular decomposition schemes used and types of geometric primitives permitted, the resulting trees can have different shapes and the contents of intermediate nodes vary depending on whether or not geometric elements participate in the

partition. The leaves of a tree can contain a single geometric object or a cell of multiple objects.

The idea of geometric hierarchical structures starts with the kd-tree [Bentley 1975]. A kd-tree stores k-dimensional points. In two dimensional cases, the root of the tree corresponds to the whole region of interest (Figure 2.21). The rectangular region is divided into two parts by the x-coordinate of the stored point on the odd nodes and by the y-coordinate on the even nodes on the tree. This division process continues recursively until a leaf node is reached on which no more than one point resides. This structure guarantees a logarithmic time for an exact match search. The original kd-tree manages points only and it does not consider disk paging capability. The balance of the kd-tree largely depends on the choice of the root node, which further requires that all data be ready before the tree is constructed. Regular balance operations may be applied as points are inserted or deleted.

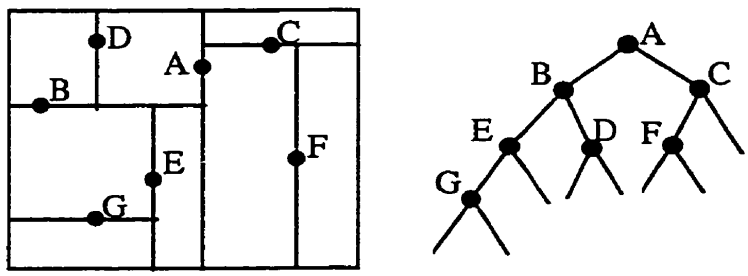


Figure 2.21 A 2D-tree decomposition of space

Variants of the original kd-tree, as well as many other types of trees, have been developed to accommodate complex objects and to enable secondary storage. For example, Matsuyama et al. [1984] modifies the kd-tree to index axes-parallel rectangular blocks where points, lines and polygons are recorded. The PM Quadtrees [Rosenberg 1985; Samet 1984], which are regulated to have four branches, are used to partition a polygonal map. The arc tree [Günther and Wong 1989] generalizes arbitrary curved shapes. Nevertheless, all these trees divide complex objects into subobjects at the boundaries of partitioned cells. The fragmentation of geometric objects makes spatial queries such as point-in-polygon queries difficult and inefficient. Generally, it reduces the update dynamics and requires

increased effort to balance the structure when deletions and additions occur. We make a note here that a true deletion with this kind of tree structures is really impossible. An example of a PM Quadtree is given in Figure 2.22.

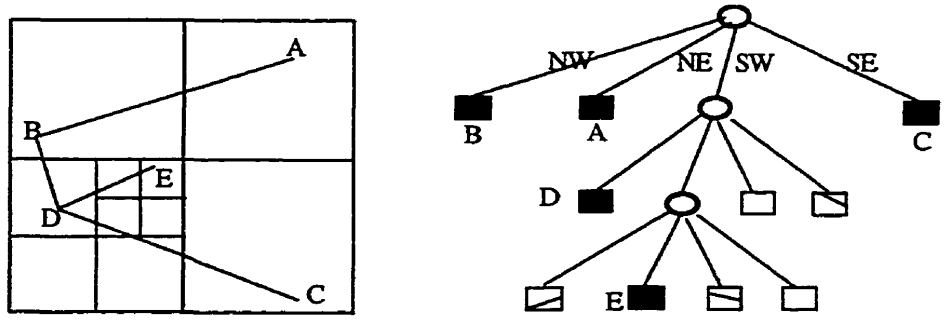


Figure 2.22 A PM Quadtree decomposition of space

The R-tree [Guttman 1984] is a multi-dimensional extension of the B-tree and can be applied to accommodate complex objects with no fragmentation. In an R-tree (Figure 2.23), each node represents a rectangle which can correspond to a disk page. The leaf nodes are the containers that actually hold the data in disk storage. Guttman devised algorithms for inserting and deleting operations which ensure that each node is neither overflow nor underflow for a given branching factor and that the tree is always balanced.

The major problem with the R-tree is the possible overlapping of certain rectangles which causes overheads in spatial searches. The R^+-tree [Faloutsos et al. 1987] is then developed to overcome this problem by ensuring that rectangles do not overlap. This approach, however, causes another problem: a complex object may be broken into different rectangles.

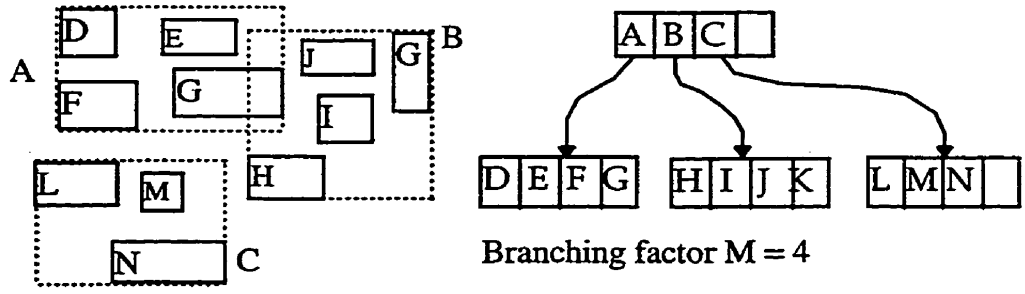


Figure 2.23 The R-tree decomposition of space

The Cell-tree (Figure 2.24) [Günther 1988] is designed to facilitate searches on polygonal objects. Each cell node corresponds to a convex polygon which is the result of binary space partition (BSP). The leaf nodes, which are all on the same level, contain all information that may be required to answer a query. Each node of the Cell-tree corresponds to one disk page. Günther used a tree condensation operation to eliminate empty leaves and propagates the elimination up the tree. Interior nodes with the number of entries less than the minimum are deleted and the entries under these nodes are reinserted into the cell tree. The shortcomings of the Cell-tree include: 1) the definition of a tree node object does not corresponds to a native object definition, for not all polygonal objects are convex; 2) in order to split a region into convex polygons, native objects may have to be broken into different tree nodes; 3) splitting convex polygons is more complicated than partitioning axes-parallel rectangles as practiced by R-trees.

The Field-tree [Frank 1983; Frank and Barrera 1989] has common features with the Grid file and the PR quadtree. It stores complete objects (can be polylines and polygons) in rectangle cells each of which belongs to a field partitioned at a certain resolution level. A Field-tree therefore may consist of several sets of grids at different resolutions and displacements. An object may be inserted into a grid cell if it does not overlap the grid cell boundaries and there is no finer-meshed grid that will hold the object. An object can also be stored in a higher level field if it is considered more important. A drawback to the Field-tree is that overflow pages are sometimes required when grid cells of one field becomes full. We note here that the ability to manipulate spatial objects according to their thematic and geometric significance is a desirable feature in the design of a GIS, although it might

be inappropriate to enable this feature in the geometric data structure. Some of the reasons for this argument are mentioned below and will be elaborated when we discuss problems in automated map generalization.

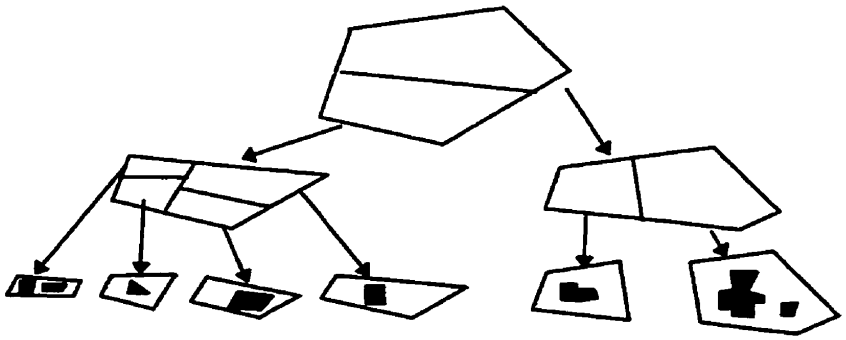


Figure 2.24 A Cell-tree decomposition of space

Another geometric data structure that stores and retrieves geometric objects at different levels of detail is described by Oosterom [1993]. He defined this kind of scheme as the *reactive data structure*. A reactive data structure is closely related to cartographic map generalization concepts and techniques. In the reactive data structure, a geometric reactive-tree must be constructed with embedded object-selection mechanisms. As with the Field-tree, the reactive-tree assumes that some important values are associated with geometric objects. One guideline for generating the reactive-tree could be: important objects are stored in the higher levels of the tree. Associative structures might be added to support different aspects of the generalization process. A general reactive-tree is a multi-way tree in which each internal node contains both object and tree type entries. The leaf nodes contain object-entries only. An object-entry has the form

(MBR, imp-value, object-id)

where MBR is the minimum bounding rectangle, imp-value is an integer indicating the importance of the object, and object-id contains a reference to the object. A tree-entry has the form

(MBR, imp-value, child-pointer)

where child-pointer contains a reference to a subtree. The MBR in a tree-entry is the minimum bounding rectangle of the whole subtree and imp-value is the importance of the child-node incremented by 1. Both entry types have the same size, with one bit in the object-id/child-pointer to discriminate between the two entry types. Each node of the reactive-tree corresponds to one disk page.

The reactive-tree satisfies the following properties:

- For each object-entry (MBR, imp-value, object-id), MBR is the smallest axes-parallel rectangle that geometrically contains the represented object of importance imp-value.
- For each tree-entry (MBR, imp-value, child-pointer), MBR is the smallest axes-parallel rectangle that geometrically contains all rectangle in the child node and imp-value is the imp-value of the child-node incremented by 1.
- All the entries contained in nodes on the same level are of equal importance, and more important entries are stored at higher levels.
- Every node contains between m and M object-entries and/or tree-entries, unless it has no brothers (a pseudo-root).
- The root contains at least two entries, unless it is a leaf.

Figure 2.25 shows a set of objects: objects of importance 1 are white, and those of importance 2 are gray. The figure also shows the corresponding rectangles as used in the reactive-tree. The object-entries are marked with a circle in Figure 2.26 for this example.

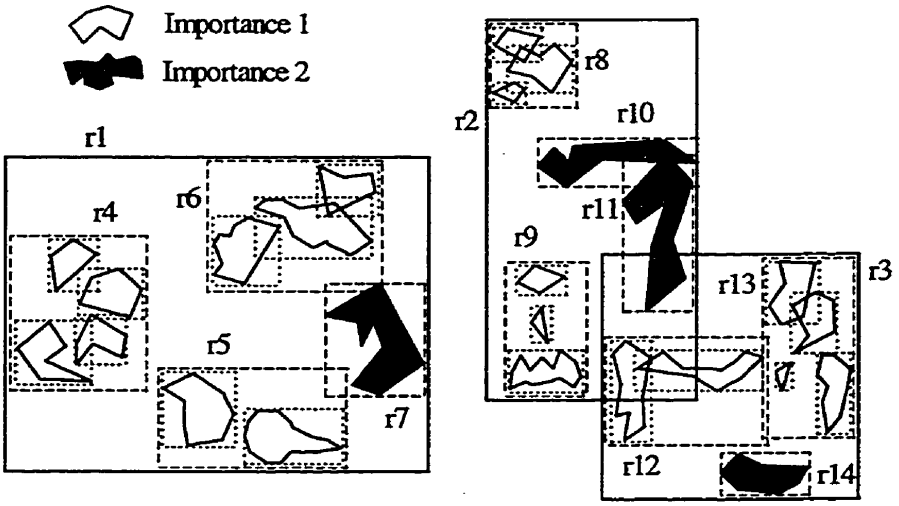


Figure 2.25 Objects and containing rectangles of the reactive data structure

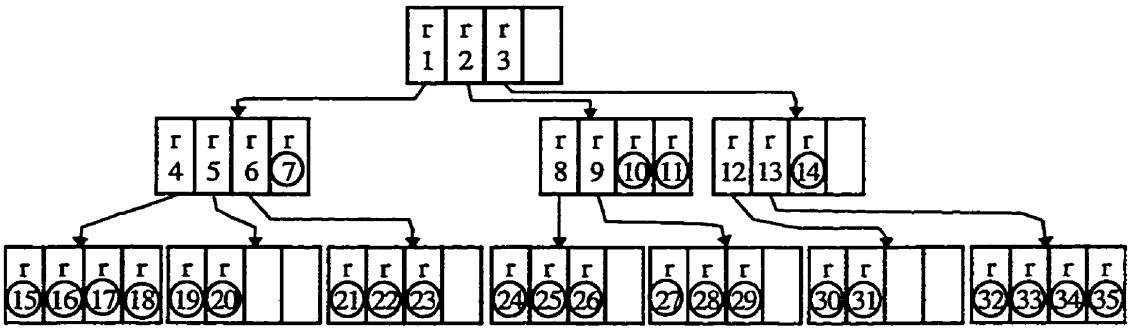


Figure 2.26 The Reactive-tree for the configuration in Figure 2.25

The drawbacks of the reactive-tree are: First, it imposes the knowledge of importance values of objects resulted from the map generalization techniques before the tree can be constructed. Classifying spatial objects into different important groups is strongly application dependent. While the resulting tree is efficiently targeted to a particular application, it may not be a geometric data structure supporting general applications. Besides, the data input process itself becomes application oriented. Digitized map data categorized solely by their types of geometric primitives cannot be directly used to build the tree. Second, as with most of the tree indexing structures, inserting and deleting objects may cause severe imbalances at the lower levels of the tree. Furthermore, a lower level node split may trigger overflow in an index node above, which in turn may trigger a further

cascade of downward splits. Third, as with other geometric data structures, the reactive structure does not contain topology of the spatial objects. Topological queries have to be supported by constructing associated topological data structures. The separation of the geometry and topology into different indexing structures degrades the integration of the system and makes the maintenance of the two indices difficult.

In summary, the geometric data structures that support complex objects either cut objects into pieces which may be stored in different disk pages or create overlapped containers which causes inefficiency in spatial searches. Although most of the structures have disk paging ability, the resulting disk pages themselves are not meaningful objects. They serve simply as a part of the data store but cannot be worked upon independent of the whole indexing scheme. The dynamics vary from one geometric data structure to another and are enabled at the expense of balancing operations or additional overflow handling. All geometric data structures do not explicitly consider the construction of the topological relationships. The concern is focused only on handing space partition problems.

2.5 Problems with Current Spatial Database Models

Most current spatial databases in GIS use a hybrid architecture. That is, the topology and geometry about spatial data are modelled separately with topological and geometric data models. Figure 2.27 illustrates the relationships and levels of representations between the two data models. In the diagram, a continuous space is firstly discretized and spatial objects are collected (identified with data types, names, and geometry). The storage of spatial objects is managed by a geometric data model. The geometric data model dispatches spatial objects into decomposed cells which are possibly grouped into disk pages. Based on the geometry stored, the topological relationships between spatial objects (identified with data types and names) are then extracted and represented based on either planar graphs or simplicial complexes. The storage of the representation may utilize a relational DBMS or specialized file structures.

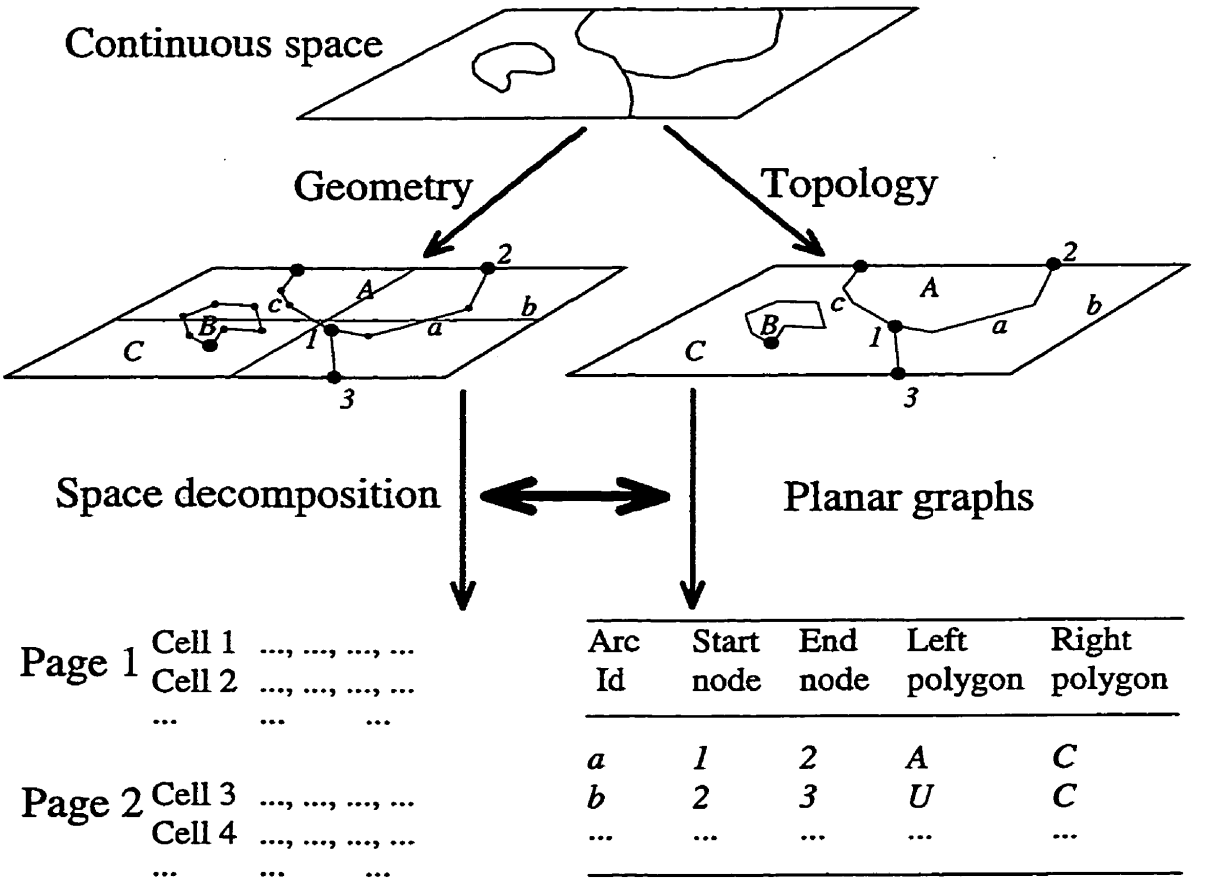


Figure 2.27 The hybrid architecture of spatial database models

The drawbacks of the hybrid architecture to manage spatial objects can be addressed from the following points of view:

First, the two data models have different mathematical bases. The geometry data model is based on the decomposition of the space. The geometric reference framework resulting from the decomposition of the space is a rather rigid one which does not generally care about the extent, shape, and complexity of the embedded objects. As a consequence, some complex spatial objects have to be broken into pieces, the number of which depends on the partitioning criteria used. The fragmentation induces extra costs for maintaining identity, and reduces the integrity of spatial objects. The dynamics of the geometric structure is also reduced due to the fragmentation. The main purpose of using a geometric data structure is to form indices for the storage of clusters of

geometric data. The topological relationships are of little concern with a geometric scheme of organizing spatial data. Although it can be argued that some geometric structures preserve certain topological relationships in their subdivision, e.g. containment in R-trees, they are not designed to efficiently answer topological queries.

The topological data model, on the other hand, is based on a planar graph, which is an abstract data made with little handling of the geometry of spatial objects. The purpose of using a topological data model is to encode and to answer queries concerning topological relationships between spatial objects, while whose positional data are off-line with respect to the system managing the topology. The representation of a topological structure depends on a complete, flat exposition of spatial objects in the plane. Although the geometry of spatial objects may be partitioned, paged, and managed hierarchically, the topological representation cannot conform to the geometric partitioning scheme.

Second, the interaction between the two data models is not dynamic and integrated. This is especially true when modification of the spatial database occurs. Adding an new object into the database, for example, involves inserting the object first into the geometric data structure. This process creates the identifier of the new object and possibly identifiers of other system-generated objects, when the new object intersects others in the database. All these object identifiers then have to be incorporated into the topological structure and its representation. The modification of the topological data structure generally needs to calculate, for each new object, the incidence and adjacency with other objects, and to insert new relations into the data structure. The catch is that existing topological relations may be altered with the insertion. Failure to detect and accommodate all these changes causes topological errors. Most current GIS systems therefore ask the operator to execute an exhaustive rebuilding process once the topology of the database is changed, although some tactic operations may reduce this probelm. This analysis implies that the linkage between the two data models is loose and apt to inconsistency.

Third, it is known that the planar graph topological model lacks the power of expressing some topological relationships. The representation of the topological structure relies on the incidence relationship between spatial objects. It is weak when addressing the containment topology with independent points and disconnected components in a larger embedding space. Even for the connected components, the data model has difficulty distinguishing and encoding the nature of the connection (weak or strong). As is mentioned in the first point, the data model lacks the power of representing spatial relations at different levels of abstraction, which tarnishes the abstracting effort made with hierarchical representations during conceptual and geometric modelling processes. On the other hand, representing spatial objects and relationships at different levels of detail is natural and is the trend for spatial data modelling.

Finally, from the systems point of view, the separation of geometry and topology representations of spatial objects is not supportive of persistent object-oriented modelling technology. It is impossible to define classes of spatial objects which encapsulate their states, spatial properties, and operational functions. The only topologically operable object type is the map (the coverage within Arc/Info™ terminology) itself which constitutes the contents of the whole spatial database. The manipulating of lower level spatial objects must be performed through procedures (DML) provided by the spatial DBMS. It is understood that these procedures are coded separately from the storage of data to be accessed. When a map is accessed and manipulated, other accesses to portions of the database will be refused. Likewise, the communication of the spatial data can only occur at the map level if both their geometric and topological structures are required. A true realization of the client-server architecture would be difficult with a non object-oriented spatial database model, especially when frequent modifications of a map are needed.

Chapter 3

The Dynamic Voronoi Data Model

A spatial data model serves to represent and manage spatial objects occupying pieces of a space. If there were no embedded spatial objects then the space is homogeneous and there is no reason to manage it. The fact is that we do have various kinds of spatial objects which dot our modelling space! At the beginning of this chapter, let us come back a bit to the dichotomous views of modelling a geographical space to see if we can combine both views within one integrated modelling system.

3.1 An Integrated View of Modelling Space

Imagine that we are modelling the ecosystem around each tree in a forest within the Euclidean plane (Figure 3.1). The vicinity (the gray circle) centred on each tree (the black dot) has an influence on the growth cycles of its flora and fauna. The influence is greater nearer the tree and weaker further out. At some point, part of the boundary of the expanding vicinity will meet with other vicinities centred at and extending from neighbouring trees. Those boundaries that do not meet will extend to the edge of the space (Figure 3.2). Assume that all trees in the forest have similar size and that the expanding speed is the same for each vicinity in all directions. The meeting boundaries will have the same distance from each of the neighbouring trees. For any point within a vicinity, the distance between the point and the tree will always be shorter than the distance between this point and any other tree.

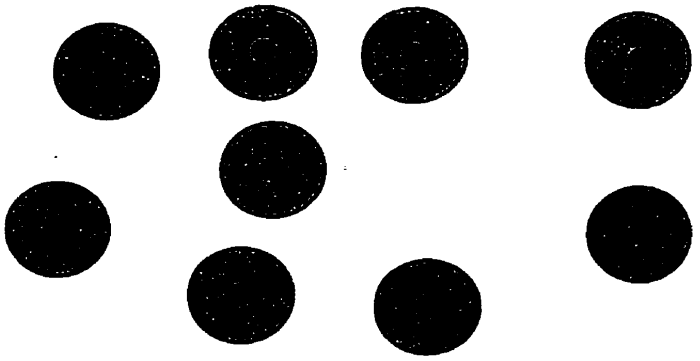


Figure 3.1 Trees and their vicinity circles

Figure 3.2 The tessellated space with respect to trees

Figure 3.2 forms a tessellation of the space with respect to the distribution of trees. In other words, the space is partitioned to subspaces or tiles each of which is associated with a single tree within it. It is this tessellation that combines both field-based and object-based views of spatial data models. From the field-based point of view, each tile represents collectively the locations that influence or are influenced by the growth of the associated tree. Across the tile boundary will be an area dominated by one of its neighbouring trees. From the object-based point of view, the reason for the tessellation of the space is the existence of those trees without which the modelling space will not be tessellated. The tessellation model, therefore, is centred on the objects and is simultaneously expressed by the collection of locations surrounding each object. With this integrated view, when one

speaks of an object in the tessellation, there is a tile associated with it. In reverse, when one refers to any location within a tile, the object dominating this tile is known.

Besides tessellating a space with respect to points, we can similarly have a tessellation with respect to polylines and polygons (Figure 3.3). The analogy is the same as that for points: tiles are formed by expanding their vicinities in all directions from each complex object until all boundaries either meet within the space or extend to infinity. The collection of tiles is a partition of the space and each tile is associated with an object within it. Any location within a tile is closer to the associated object than to any other object.

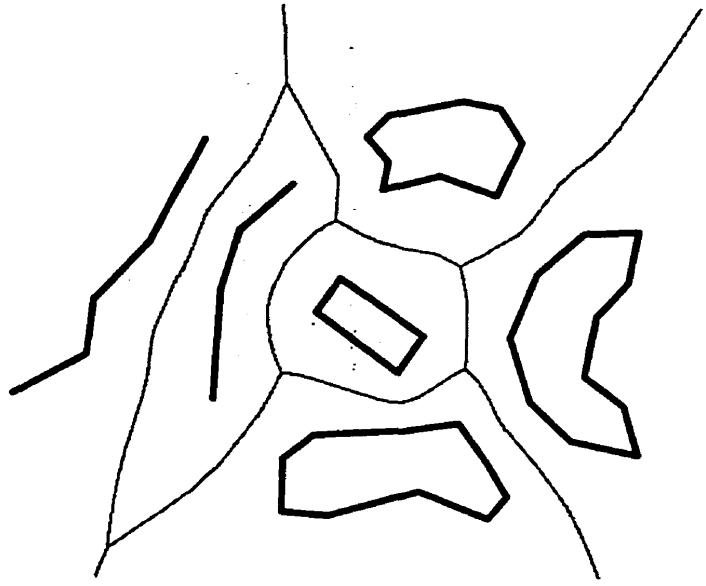


Figure 3.3 The tessellated space with respect to polylines and polygons

The tessellation of a space based on the vicinity analogy has a special name, called the Voronoi diagram, which will be formally defined in the next section. Figure 3.2 is a Voronoi diagram of points. Figure 3.3 is generalized from a Voronoi diagram of points and line segments. One can draw or sketch such Voronoi diagrams on a piece of paper without much difficulty. We will see in this chapter how they can be produced with a computer.

What is so special about the tessellation of a space via a Voronoi diagram? Tessellating a space is not really new. All the geometric structures of spatial objects that we have seen are

tessellations in one form or another. Linear orderings tessellate a space into stripes of consecutive spatial units; bucket structures divide a space into rectangular cells based on a grid framework, although the size of buckets can vary due to the merging of cells; tree structures tessellate first into two, four, or any limited number of containers, and repeat this process in lower level containers until the contents of the container reach a pre-defined criterion. The purpose of all tessellations is the same: partition a space into smaller ones such that they are easier to manage. The key differences for the Voronoi diagram are:

1) The Voronoi tessellation is always object-based in that no tiles of space exist if there is no object embedded in the space.
2) The tessellation is also object-driven in that if there is an object, there is a tile associated with it; on the other hand, the tile disappears when the object is deleted from the space.
3) No additional breaking points will occur in the partitioning, in that a single identified object never belongs to two tiles.
4) The tessellation is irregular depending on the distribution and on the geometric configuration of objects; distribution and configuration of objects are not generally uniform.

3.2 A Formal Definition of Ordinary Voronoi Diagrams

In this section we define Voronoi diagrams. Of the numerous types of Voronoi diagrams, we are especially interested in the *ordinary* Voronoi diagram. This kind of Voronoi diagrams is "ordinary" because it is simple and follows the intuition of the vicinity analogy. Ordinary Voronoi diagrams can be defined with a single notion of the Euclidean distance metric. It turns out that varying the distance function alone tells a lot about the behaviour of Voronoi diagrams. Besides the Euclidean metric, non-Euclidean metrics can be used to define Voronoi diagrams with special properties, which find interesting applications. Extensive and scholarly surveys on both ordinary and "unordinary" Voronoi diagrams are provided by Aurenhammer [1991] and Okabe et al. [1992].

We consider first a set of points $S = \{s_1, s_2, ..., s_n\}$, for a limited integer n ($n > 1$), embedded in the R^2 Euclidean space such that no two points coincide. The ordinary *Voronoi diagram*, $V(S)$, partitions the plane into *Voronoi regions*, v, such that for two distinct points $s_i, s_j \in S$, the dominance of s_i over s_j is defined as the subset of the plane being at least as close to s_i as to s_j :

$$D(s_i, s_j) = \{x \in R^2 \mid d(x, s_i) \leq d(x, s_j)\} \qquad (3.1)$$

where d denotes the Euclidean distance function. $D(s_i, s_j)$ is a closed half plane bounded by the bisector of s_i and s_j, denoted by $B(s_i, s_j)$:

$$B(s_i, s_j) = \partial D(s_i, s_j) = \{ x \in R^2 \mid d(x, s_i) = d(x, s_j)\} \quad (3.2)$$

The Voronoi region $v(s_i)$ is the portion of the plane lying in all of the dominances of s_i over the remaining points in S:

$$v(s_i) = \bigcap_{s_i \in S - \{s_j\}} D(s_i , s_j) \qquad (3.3)$$

With these, the Voronoi diagram $V(S)$ is finally expressed as the collection of Voronoi regions:

$$V(S) = \{v(s_i) \mid s_i \in S\} \qquad (3.4)$$

If two bisectors $B(s_i, s_j)$ and $B(s_i, s_k)$ (s_i, s_j, and $s_k \in S$) intersect, the intersection is called a *Voronoi vertex* and the bisector, delimited by two consecutive Voronoi vertices, is called a *Voronoi edge*. We denote $Q = \{q_i\}$, for $1 \leq i \leq n_q < \infty$, be the set of Voronoi vertices of a Voronoi diagram generated by S. Two points in S are *neighbours* if they share a common Voronoi edge (possibly extended to infinity). The Voronoi vertex has an equal-distance to at least three points and is therefore the circumcentre of a circumcircle, C_i ($1 \leq i \leq n_q < \infty$),

defined by these points. Figure 3.4 shows the ordinary Voronoi diagram for a set of points with a graphic depiction illustrating the elements defined in this section.

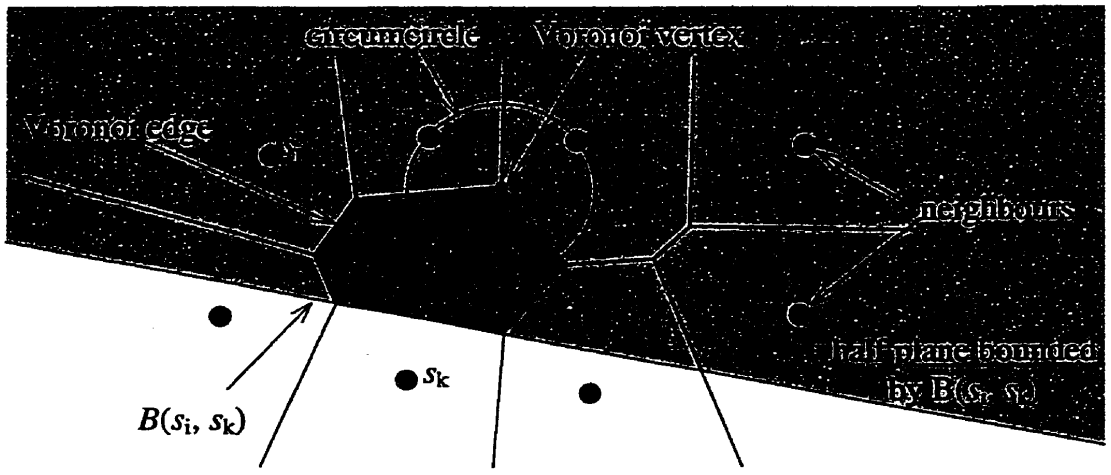


Figure 3.4 The ordinary Voronoi diagram and related elements

Next we consider the finite set $S = \{s_1, s_2, \ldots, s_n\} \subset R^2$, where $s_i \in S$ is a point, a line segment, a polyline, or a polygon as defined in Section 3.2. The above definition can be generalized to ordinary Voronoi diagrams constructed from the set S. The generalization is made with respect to the extended definition of the distance function d which has been used in the definitions of the point Voronoi diagram and related elements. Generalized Voronoi diagrams of this kind have been studied in computational geometry since the late 1970s started by Drysdale and Lee [1978] (also reported in, Drysdale [1979], Kirkpatrick [1979], Lee and Drysdale [1981]). In a generalized Voronoi diagram, the distance function d is defined as the shortest distance between a point $x \subset R^2$ and an arbitrary point x_i on $s_i \subset S$, that is

$$d_s(x, s_i) = \min_{s_i} \{\|x - x_i\| \mid x \in R^2, x_i \in s_i \subset S\} \qquad (3.5)$$

where $\boldsymbol{x}$ and $\boldsymbol{x}_i$ are the location vectors of x and x_i, respectively. For a line segment s_i with two end points s_j and s_k, the specification of the shortest distance function d_s can be written as

$$d_s(x, s_i) = \begin{cases} \|x - s_j\|, & \text{if } x \in R_{i1}, \\ \|x - s_k\|, & \text{if } x \in R_{i2}, \\ \left\| (x - s_j) - \dfrac{(x - s_j)^T (s_k - s_j)}{\|s_k - s_j\|^2} (s_k - s_j) \right\|, & \text{if } x \in R_{i3}, \end{cases} \tag{3.6}$$

where x, s_j, and s_k are location vectors of a point $x \in R^2$, and two endpoints s_j, $s_k \in S$ respectively, and

$$R_{i1} = \{x \mid x \in R^2, (s_k - s_j)^T (x - s_j) < 0\},$$
$$R_{i2} = \{x \mid x \in R^2, (s_k - s_j)^T (x - s_k) > 0\},$$
$$R_{i3} = R^2 \backslash [R_{i1} \cup R_{i2}]$$

are three regions for point x in respect to the line segment s_i (Figure 3.5).

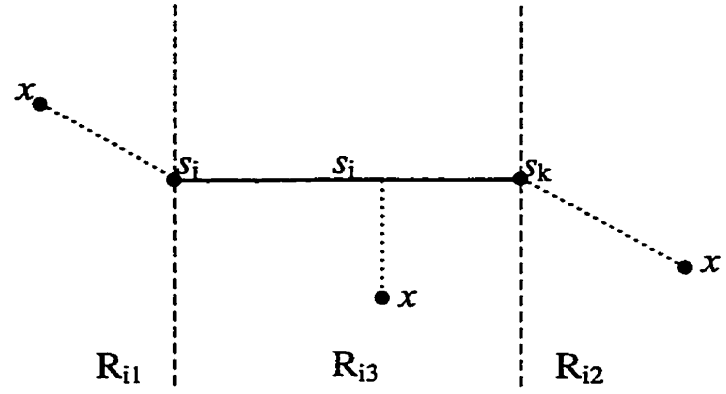


Figure 3.5 Regions for calculating the distance from a point to a line segment

Having defined the shortest distance function for a line segment (for a point as well when a line segment has degenerated into a point), the shortest distance from a point to any other complex object in S is known, since all complex objects defined for S are connected with line segments. In the next section, we explore properties of the ordinary Voronoi diagram

of points and line segments. Because we are dealing exclusively with the ordinary Voronoi diagram in this thesis, the modifier "ordinary" will be omitted from the context when no confusion arises.

3.3 Properties of the Voronoi Diagram of Points and Line Segments

The first property examined demonstrates the behaviour of Voronoi edges which form the boundary of a Voronoi region at the most primitive level.

Property 3.1. Given the object set S defined earlier for a generalized Voronoi diagram, there are four possible types of Voronoi edge which bisect: i) two points (type E1); ii) an endpoint and its connected interior of a line segment (type E2); iii) a point and the interior of a line segment (type E3); and iv) the interiors of two line segments (type E4).

The nature of the four types of Voronoi edges can be observed diagrammatically in the sketches shown in Figure 3.6.

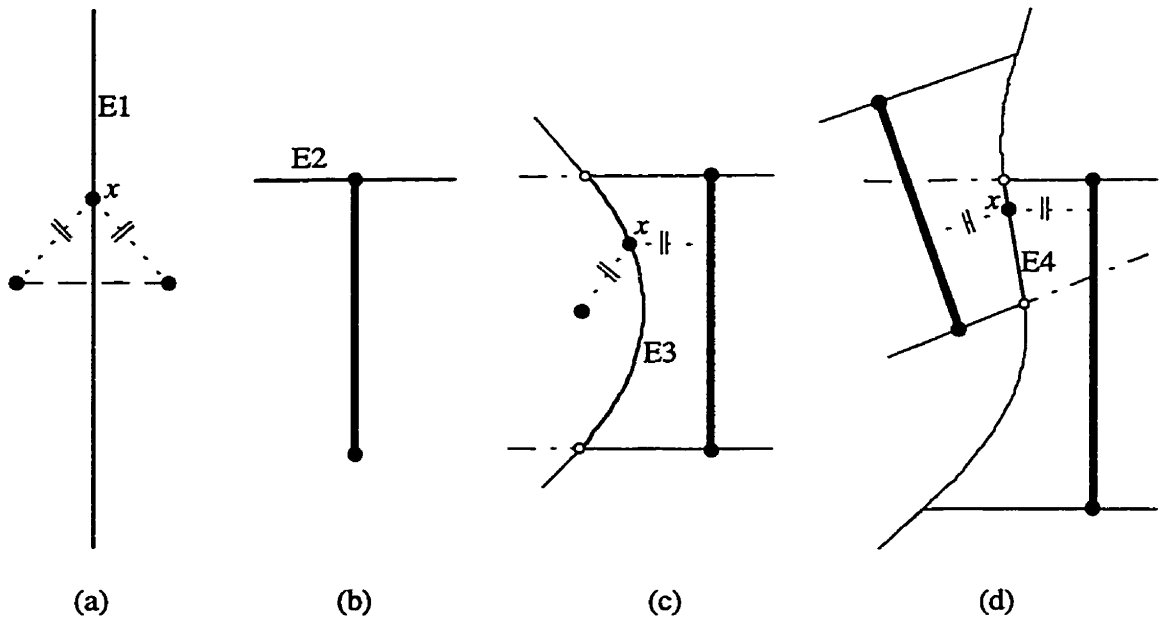


Figure 3.6 Four possible types of Voronoi edges bisecting two objects

The diagram depicts that:

A Type E1 edge (a) is a straight line and is perpendicular to the line linking the two points.

A Type E2 edge (b) is also a straight line perpendicular to the line segment and passing through the endpoint.

A Type E3 edge (c) is a simple parabolic curve and is bound within the subspace overlapped by three half-planes with respect to the line segment and its two type E2 edges. The terminating points of the type E3 edge must be Voronoi vertices (hollowed circles). Each terminating point is the centre of the circumcircle formed by the point, the line segment, and at least one other object.

A Type E4 edge (d) is a straight line, bisecting the angle between the two line segments, and is bound within the subspace overlapped by six half-planes with respect to the two line segments and their type E2 edges.

It should be emphasized here that the introduction of type E2 Voronoi edges is the consequences of 1) the definition of the line segment in S, and 2) the distance function defined by Equation 3.6. There are Voronoi diagrams of line segments without the type E2 edges (c.f. Gold et al. [1995]). These Voronoi diagrams do not recognize the endpoints as a distinct object in the defining object set.

All types of Voronoi edge are equidistant from the two neighbouring objects which define it. An example of the distance from any point x on each Voronoi edge projected onto two neighbouring objects $s_i, s_j \in S$, $i \neq j$, is given in Figure 3.6, as marked by the symbol "//". Using this fact ($d(x, s_i) = d(x, s_j)$), together with the distance functions given in Equation 3.6, one can verify the nature of the four types of Voronoi edges with analytical functions.

The existence of the E3 type Voronoi edge in the Voronoi diagram involving line segments has the following consequences which are stated as properties.

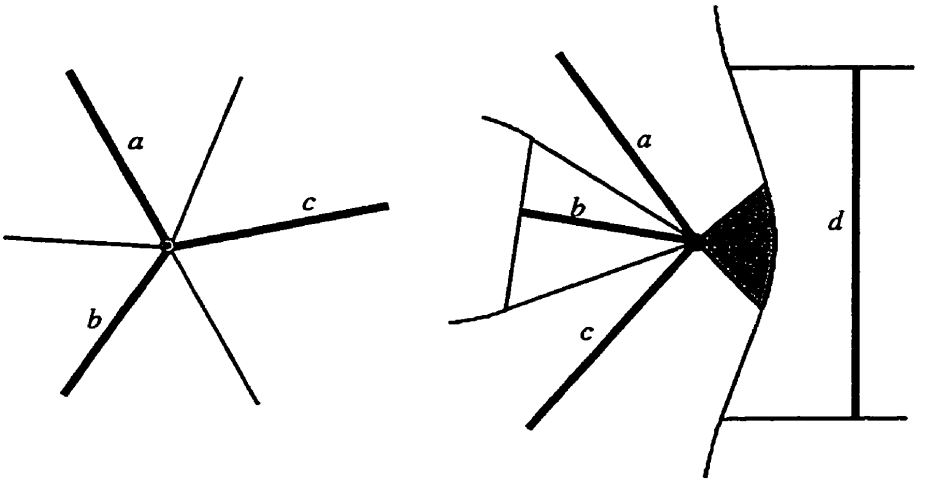
Property 3.2. A Voronoi region in the Voronoi diagram of points and line segments is not necessarily convex.

Property 3.3. A bounded Voronoi region in the Voronoi diagram of points and line segments may have only two parabolic Voronoi edges (Figure 3.7).

Figure 3.7 Voronoi regions bounded by two Voronoi edges

Property 3.4. The Voronoi region of an endpoint incident to three or more line segments may be contracted into one point (Figure 3.8a).

We note that not all the Voronoi regions of endpoints incident to three or more line segments will be contracted into a point. This is the case when polylines are connected to form more complex objects (Figure 3.8b).



a) A contracted Voronoi region of an endpoint (grayed dot)

b) A non-contracted Voronoi region of an endpoint (grayed area)

Figure 3.8 Contracted and non-contracted Voronoi regions of endpoints incident to three or more line segments

These properties do not apply to the Voronoi diagram for a set of points where all Voronoi edges are straight lines, all Voronoi regions are convex, all bounded Voronoi regions must have at least three Voronoi edges, and there are no contracted point Voronoi regions for a discrete point set. The non-convexity of the Voronoi region presents some challenge to the design of geometric algorithms which are otherwise easier when all Voronoi regions are convex.

It has been mentioned in the definition of a Voronoi diagram that the Voronoi vertex is equidistant to at least three points and is therefore the circumcentre of a circumcircle defined by these points. This fact is equally true for a Voronoi diagram of points and line segments where a line segment must be tangent to a circumcircle. The following property concerns the nature of any circumcircle defined by three or more objects.

Property 3.5 (the empty circumcircle theorem). For every Voronoi vertex, q_i, in a Voronoi diagram, there exists a unique empty circle C_i centred at q_i with contact points on three or more objects in S (tangent points on line segments).

This property is well known in the study of the Voronoi diagram. The proof is simple and can be found in the literature (e.g. [Okabe et al. 1992], pp. 81). We make use of this property to show the next property.

The property we now examine classifies possible types of Voronoi vertices $\{q_i\}$ in a Voronoi diagram. It is closely related to Property 3.1 and is useful for studying methods to calculate Voronoi vertices. In computational geometry the following two assumptions are usually made in discussing the properties of Voronoi diagrams:

The non-collinearity assumption. For a given set of objects $S = \{s_1, s_2, \ldots, s_n\} \subset R^2$, $(3 \leq n < \infty)$, $s_1, s_2, \ldots, s_n$ are not on the same line.

The reason for this assumption is clear. If all objects in S lie on the same line, all the Voronoi edges will be perpendicular to the line and extend to infinity without being intersected within a limited range. This is a special case which requires particular treatment.

The general position assumption. A given set of objects $S = \{s_1, s_2, \ldots, s_n\} \subset R^2$, $(3 \leq n < \infty)$, is in general position if no four objects are cocircular and no three objects are collinear.

Satisfying this assumption, exactly three Voronoi edges will be incident at every vortex in a Voronoi diagram. The Voronoi diagram which is compatible with this assumption is *non-degenerate*, otherwise it is *degenerate*.

Property 3.6. Voronoi vertices $\{q_i\}$ in a non-degenerate Voronoi diagram for points and line segments can be classified into six types according to the types of the three Voronoi edges which are generated by three objects.

Type V1. A vertex q_i generated by three points. In this case, the Voronoi edges are all of type E1 (Figure 3.9a).

Type V2. A vertex q_i generated by a point, the interior of a line segment, and one endpoint of the line segment. The Voronoi edges are types E1, E2, and E3. The E1 and E3 edges are tangent at the Voronoi vertex q_i (Figure 3.9b).

Type V3. A vertex q_i generated by the interior of a line segment, and two points. The Voronoi edges are types E1 and E3 (Figure 3.9c).

Type V4. A vertex q_i generated by the interior of two line segments, and one endpoint. The edges are types E2, E3 and E4. The Voronoi edges are types E1, E2, and E3. The E3 and E4 edges are tangent at the Voronoi vertex q_i (Figure 3.9d).

Type V5. A vertex q_i generated by the interior of two line segments, and one point. The Voronoi edges are types E3 and E4 (Figure 3.9e).

Type V6. A vertex q_i generated by the interior of three line segments. The Voronoi edges are all of type E4 (Figure 3.9f).

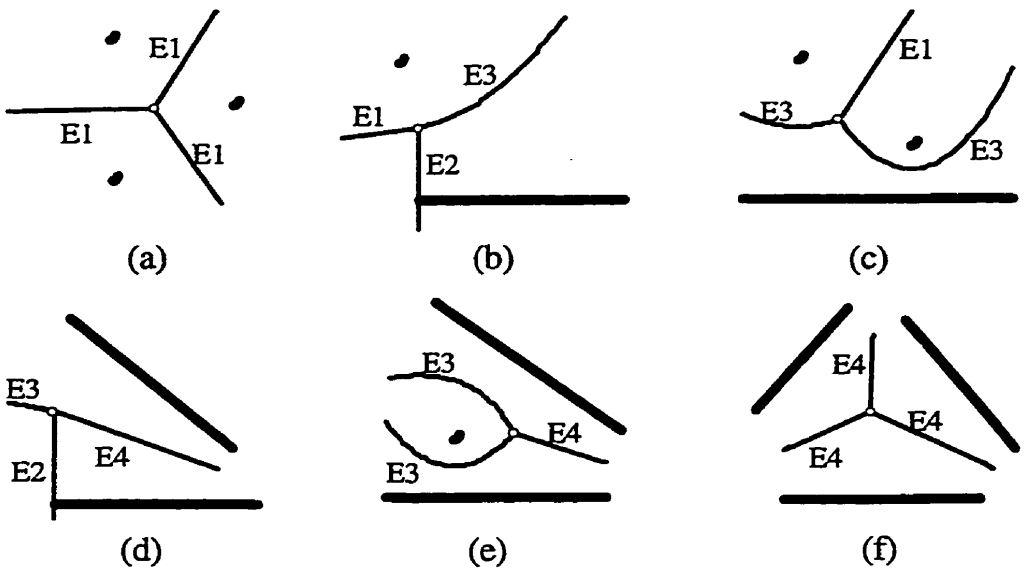
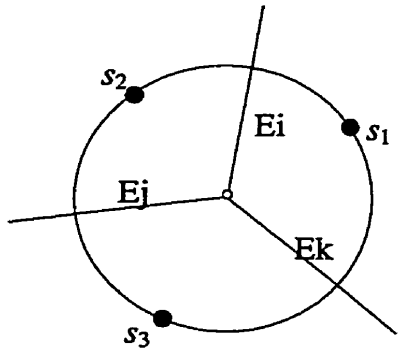


Figure 3.9 Six types of Voronoi vertices

Proof. The complete combinations of the four types of Voronoi edges (E1, E2, E3, and E4 simplified as 1, 2, 3, and 4 respectively) intersected at a Voronoi vertex q_i in a Voronoi diagram of S can be listed in four matrices (Figure 3.10a). For any configuration involving three objects s_1, s_2, and s_3 in S, three Voronoi edges Ei, Ej, and Ek (for i, j, k $\subset$ {1, 2, 3, 4}), there exist equivalent combinations (Figure 3.11). Removing surplus combinations from matrices in Figure 3.10a results in the matrices shown in Figure 3.10b.

111	112	113	114		111		
121	122	123	124			122	123	124
131	132	133	134				133	134
141	142	143	144					144
211	212	213	214		211		
221	222	223	224			222	
231	232	233	234				233	234
241	242	243	244					244
				~			
311	312	313	314		311		
321	322	323	324			322	
331	332	333	334				333	
341	342	343	344					344
411	412	413	414		411		
421	422	423	424			422	
431	432	433	434				433	
441	442	443	444					444
	(a)					(b)	

Figure 3.10 Combination matrices for three Voronoi edges



ijk ~ jki ~ kij ~ ikj ~ kji ~ jik

Figure 3.11 Equivalent combinations of Voronoi edges

We now proceed to prove that out of the remaining 20 combinations, only six of them can be found in a non-degenerate Voronoi diagram, which has only three objects on a circumcircle corresponding to a Voronoi vertex. By putting three objects on or tangent to

the circle corresponding to the types of Voronoi edges incident to the vertex, it is easy to verify that the configurations in Figure 3.9 can be achieved without violating the non-degeneracy condition. These configurations match the combinations of Voronoi edges, namely 111, 123, 133, 234, 433, and 444. However, it is impossible to achieve the other combinations in the matrices without violating the non-degeneracy condition. Figure 3.12 illustrates the proof of degeneracy of the other 14 combinations. The idea is to set up Voronoi edge configurations (solid thin lines) for the combinations with minimum number of objects on the circumcircle, and at the same time, this shows that surplus (degeneracy) Voronoi edges (dashed lines) incident to the vertex q_i necessarily incur with these configurations.

Two closely related facts will be revealed in the next section, after we introduce the triangulated dual structure of the Voronoi diagram. The first fact shows that the first six non-degenerated vertices correspond to the six distinguished, non-degenerated triangles, each triangle having three objects illustrated in Figure 3.9 as its vertices. The second fact demonstrates that all the degenerate vertices correspond to non-triangulated dual polygons, which can be decomposed into triangles distinguishable by one of the six non-degenerate triangles. The significance of identifying the six non-degenerate types of vertices is that computing a Voronoi diagram requires only six well formulated methods to calculate and verify all types of Voronoi vertices.

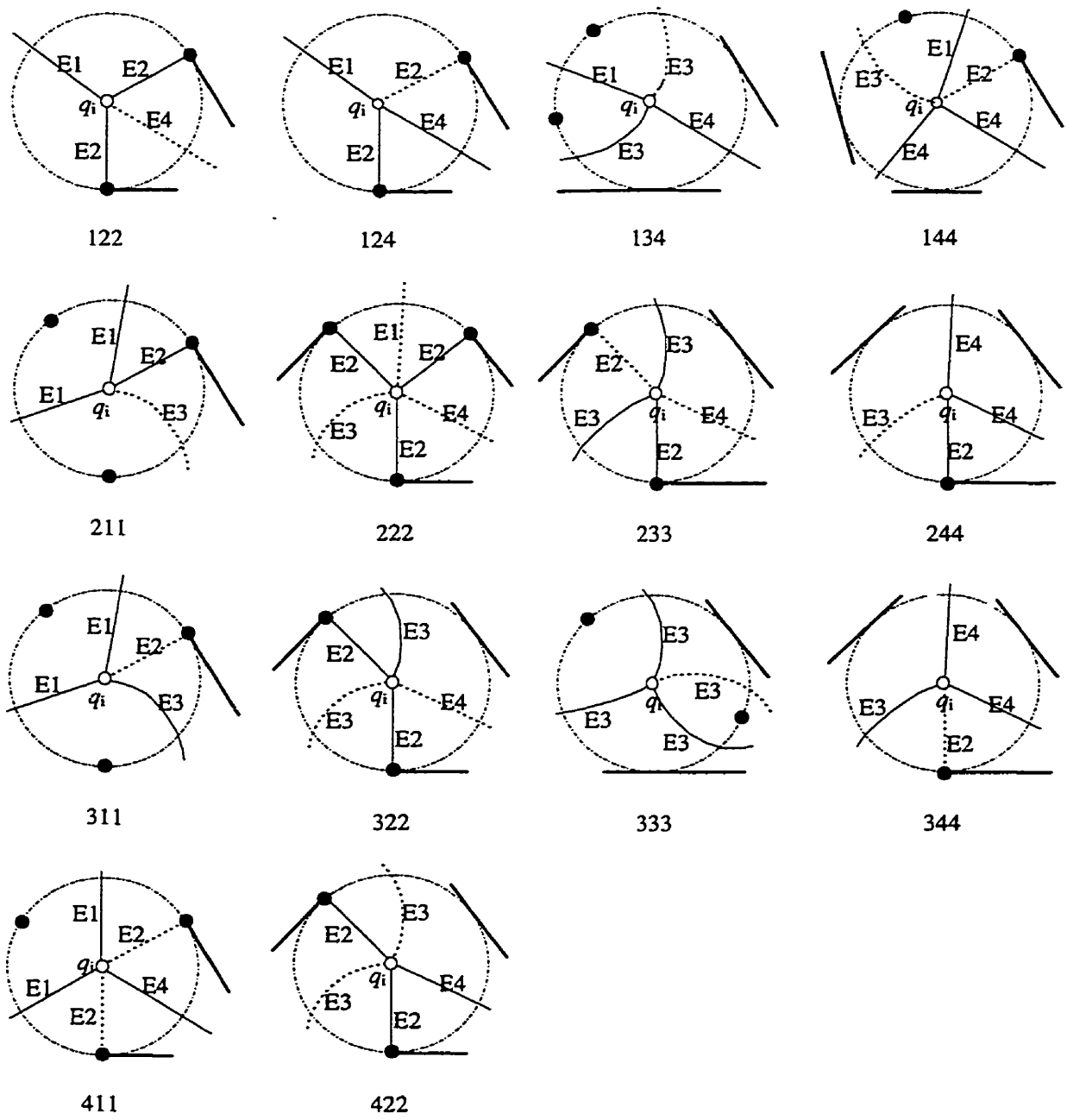


Figure 3.12 Illustration of the degeneracy of Voronoi vertices

The final property which interests us views a Voronoi diagram as a planar graph, for which the set of Voronoi vertices $\{q_i\}$ corresponds to the set of nodes, the set of Voronoi edges $\{e_i\}$ to the set of edges, and the set of Voronoi regions $\{v_i\}$ to the set of polygons. This property is stated in Aurenhammer [1991]. A proof of it can be found in Okabe et al. [1992] (pp. 83-86).

Property 3.7. For a finite set of objects $S = \{s_1, s_2, \ldots, s_n\} \subset R^2$, $(2 \leq n < \infty)$, a Voronoi diagram $V(S)$ is a planar graph $G(N, E) = G(\{q_i\}, \{e_i\})$ with the following facts:

i) There are less than $3n$ edges in the graph;
ii) There are less than $2n$ nodes in the graph; and
iii) The average number of edges of a polygon is always less than six.

This property demonstrates linear behavior for the size of the Voronoi diagram in the plane. It implies that, roughly speaking, the structure of a Voronoi diagram is not much more complex than the underlying configuration of given object set [Aurenhammer 1991]. This is one of the main reasons for the frequent use of Voronoi diagrams. A second reason is that $V(S)$ comprises the entire proximity information about S in an explicit and computationally useful manner.

3.4 The Delaunay Triangulation: The Dual Topological Structure

The graph-theoretical view of a planar Voronoi diagram permits us to examine its dual structure. To make things simple, we first study the dual structure of a non-degenerate Voronoi diagram of a set of points S. The Voronoi diagram for a set S is drawn in Figure 3.13 with thin lines. We obtain the dual structure by a *joining rule* which joins every pair of neighbouring objects with a dual edge (thick lines). Each dual edge will be orthogonal to a Voronoi edge, but not necessarily intersect it. Since each Voronoi vertex is incident to exactly three Voronoi edges, it has correspondingly exactly three dual edges which form a triangle. Voronoi regions correspond to the set of objects S. The set S equally forms the set of nodes of the dual structure. It is also observed that the boundary of the convex hull of the object set consists of dual edges. This dual structure itself constitutes another tessellation of the bounded convex region spanning the object set, and is known as the *Delaunay tessellation* in honor one of its earlier investigators, or the *Delaunay triangulation* as each tile is a triangle. A dual edge joining a pair of neighbours is called a *Delaunay edge*. Each

triangle in the triangulation is called a *Delaunay triangle* and the vertices of a Delaunay triangle, which are three points in set S, are *Delaunay vertices*. The duality between Voronoi and Delaunay immediately implies upper boundaries of $3n$ and of $2n$ on the number of Delaunay edges and triangles.

Before we give a mathematical definition of the Delaunay triangulation, a few questions addressing practical problems are worth asking:

1. The first question concerns the general position assumption. If a point set is not in general position, can we still achieve a triangulated dual tessellation for the Voronoi diagram of this set? The answer to this question is yes if the set satisfies the non-collinearity assumption. However, an additional *refined joining rule* must be observed. To see how this can be done, let us examine two degenerate Voronoi diagrams for small sets of points not in general position (Figure 3.14a and Figure 3.14d), which show that the Voronoi vertex (the small circle) in each of the two diagrams is incident by more than three Voronoi edges (thin lines).

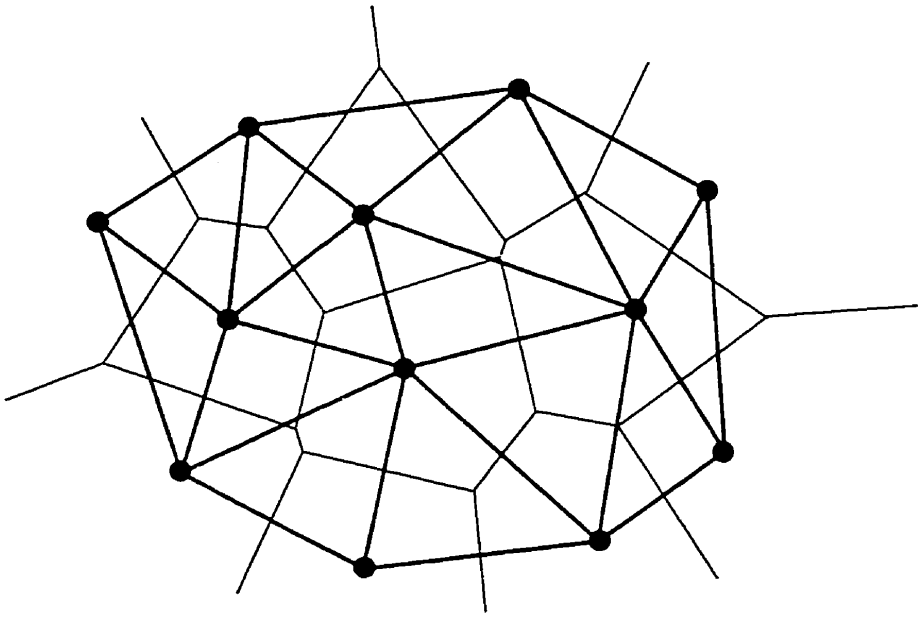


Figure 3.13 The Delaunay triangulation as a dual tessellation of the Voronoi diagram

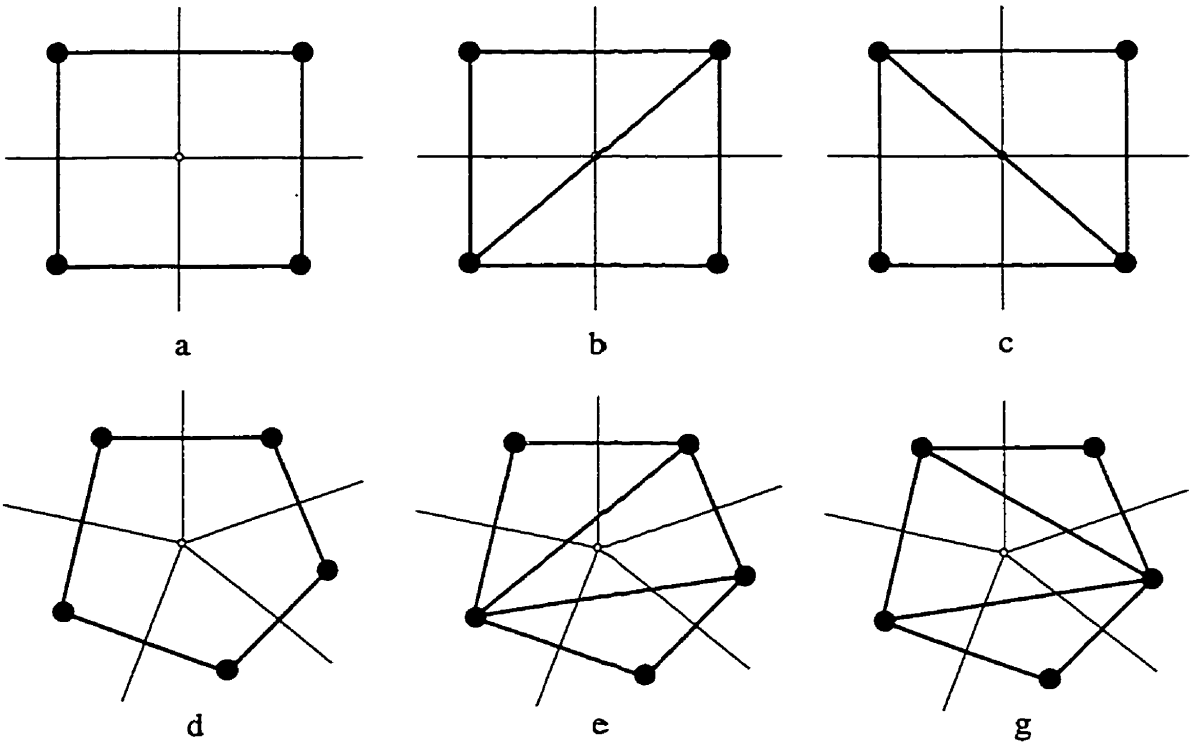


Figure 3.14 Voronoi diagrams and Delaunay triangulation for degenerated point sets

Figure 3.14a and Figure 3.14d also show dual edges (thick lines) drawn according to the joining rule to join two neighbours. They are orthogonal to the corresponding Voronoi edges. The resulting dual structures for these two Voronoi diagrams, however, are not triangulations but polygons. The refined joining rule is introduced in this case by partitioning the polygons into triangles with non-intersecting line segments joining the vertices (Figure 3.14b and Figure 3.14e). The results are therefore the Delaunay triangulations, sometimes called the *degenerate Delaunay triangulations*. Notice that a degenerate Delaunay triangulation may be structured differently by joining different objects. This can be seen from alternative triangulations (c) and (g) for (a) and (d), respectively, in Figure 3.14. Either triangulation is acceptable.

It is clear that each Delaunay triangle corresponds to a Voronoi vertex and the circumcircle defined by a Delaunay triangle satisfies the empty circumcircle theorem. Sibson [1977]

proved that the Delaunay triangulation for a set of points is *locally equiangular* or it satisfies the *local max-min angle criterion*. That is, for any two triangles whose union is a convex quadrilateral, the minimum angle among the six angles in the quadrilateral is maximized. Actually, the Delaunay triangulation is the only triangulation with this property. The Delaunay triangulation maximizes the minimum angle over all possible triangulations of a given set of points. On the other hand, a simple example shows that the maximum angle is not minimized. By the empty circle property, any triangulation without obtuse angles must be Delaunay. Triangulations without "extreme" angles are desirable in finite elements and interpolation methods.

The Delaunay triangulation is a supergraph of several well-known and widely used planar graphs spanned by a set of objects in the plane: the *minimum spanning tree* [Kruskal 1956] (or *Prim shortest connection network* [Prim 1957]); the *Gabriel graph* [Gabriel and Sokal 1969]; and the *relative neighbourhood graph* [Toussaint 1980]. These planar graphs are subjects studied in computational geometry and are applied to a variety of applications.

2. The second question asks about the extensibility of the triangulated dual structure to the Voronoi diagram constructed from a set of points and line segments. The joining rule is problematic here because a pair of neighbours may be line segments which are defined not by a single point, but a set of points defined by a linear combinatory equation. Therefore, any point on a line segment is, in theory, a neighbour to other objects in neighbouring Voronoi regions. The solution to this problem again utilizes more rules. For a line segment defined by two endpoints a and b, with $\boldsymbol{a}$ and $\boldsymbol{b}$ being their location vectors, respectively, we define the middle point located at $(\boldsymbol{a} + \boldsymbol{b})/2$ as the *graphical representation point* of the line segment. With this reinforcement, a dual edge can now be drawn to relate two neighbouring line segments by their graphical representation points. Figure 3.15 illustrates the dual triangulation derived from the Voronoi diagram of a small set of points and line segments.

Several notable characteristics of the dual triangulation can be identified in Figure 3.15. Firstly, the triangulation does not necessarily satisfy the max-min angle criterion. This is

because the triangulation simply uses the graphical representation point to show the linkage between a line segment and its neighbours. The neighbourhood relationship actually exists for all points on the line segment. Alternatively, this implies that the triangulation is not a tessellation of a space in the geometric sense. It becomes a planar graph modelling neighbourhood relationships observed from the dual Voronoi diagram. Secondly, the triangulation does satisfy the empty circumcircle theorem when the interior of a line segment is considered wholly as one vertex of a triangle. A line segment as a vertex of a triangle is tangent to the circumcircle defined by this triangle. It does not matter where the tangent point is. For example, the triangle $\Delta s_i s_j s_k$ in Figure 3.15 is corresponding to the Voronoi vertex q and has two line segment vertices s_j and s_k, and one point vertex s_i. The circumcircle C defined by this triangle is tangent to one point in each of the interiors of s_j and s_k, respectively, and is object free in its interior. For this reason, that the dual triangulation preserves the empty circumcircle property, we still call it a Delaunay triangulation. Finally, for the same reason, a triangle edge in the dual triangulation of a Voronoi diagram of points and line segments is not necessarily drawn as a straight line segment. A free curved triangle edge, t, for instance, is graphically presented in Figure 3.15. Note that the curved triangle edge is for graphical purpose only. Neighbourhood between two objects is not affected by the shape of a linking edge. The last characteristic actually reveals an important concept about the Voronoi diagram of points and line segments, that is, a line segment must be distinguished by its sides. This concept will be elaborated later.

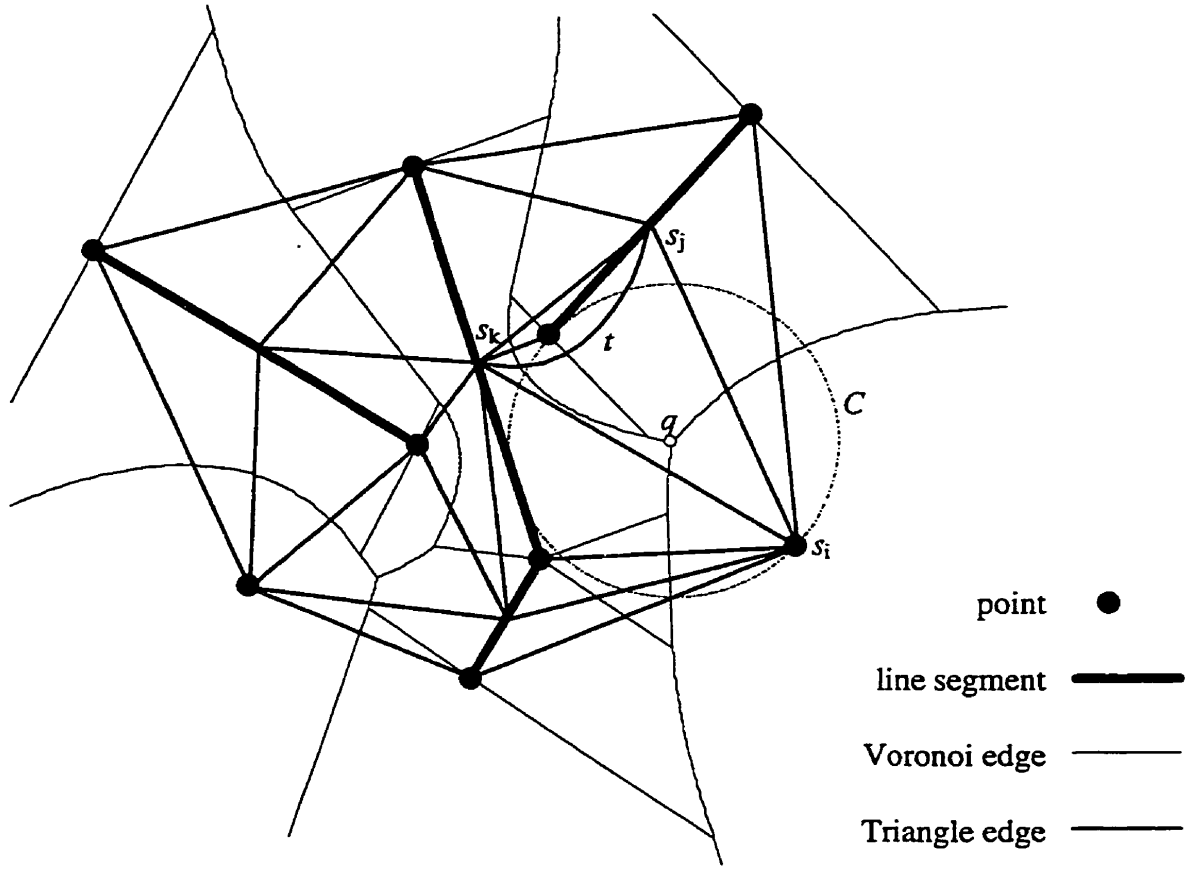


Figure 3.15 The dual triangulation of the Voronoi diagram of points and line segments

3. The last question concerns the usefulness of the dual triangulation. As the Voronoi diagram itself forms the tessellation of a space with many useful properties, what is the advantage of having a dual structure in the first place? The answer to this question emphasizes the efficiency of the dual triangulation in the representation by a computer. Similar to the topological structure discussed in the previous chapter, the dual triangulation is the topological structure which explicitly preserves the neighbourhood relationship between Voronoi tiles and hence objects. Although the Voronoi diagram itself admits this relationship, its representation by a computer is not easy. The major difficulties arise from 1) the number of neighbours of a Voronoi region depends upon the distribution of the set of objects and cannot be fixed; and 2) there are parabolic boundaries between neighbouring Voronoi regions when line segments are involved in the object set. On the other hand, the information items needed to register the same

topology can be fixed with the triangulated dual structure. In the triangulation, each triangle edge is defined by two objects in the given set, and each triangle has exactly three neighbouring triangles. Therefore, the record length for each triangle is known and constant. If the dual information (a triangle edge vs. two vertex objects and two neighbouring triangles) can be encoded in a data structure, all the facts that we need to know about a Voronoi diagram can be provided by the data structure. As a matter of fact, the dual topological data structure contains not only the explicit neighbourhood relationship, but other topological properties, such as connectivity and containment concerning topological spaces, can be derived rather efficiently. We will elaborate this point in the coming sections.

As a summary of this section, we give our formal definition of the Delaunay triangulation derived from a Voronoi diagram:

Given a Voronoi diagram for a set $S = \{s_1, s_2, \ldots, s_n\} \subset R^2$, $(3 \leq n < \infty)$, of points and line segments as defined previously, $V(S)$, its dual topological structure, the Delaunay tessellation, $D(S)$, is a planar graph $G(S, E)$, where S, E are the node set and the edge set of G, respectively, and for $s_i, s_j \in S$, $i, j \in I_n$, $i \neq j$, $s_i s_j \in E$ iff the Voronoi regions $v(s_i)$ and $v(s_j)$, possibly unbounded, share a common Voronoi edge which can be a half-line, a line segment, a parabolic curve, or a degenerate point. We call each $s_i s_j \in E$ a Delaunay edge. In the case that a Voronoi edge has degenerated into a point, this point must be a circumcentre of a circumcircle defined by more than three nodes in G, and one must therefore add Delaunay edges in E by applying the refined joining rule.

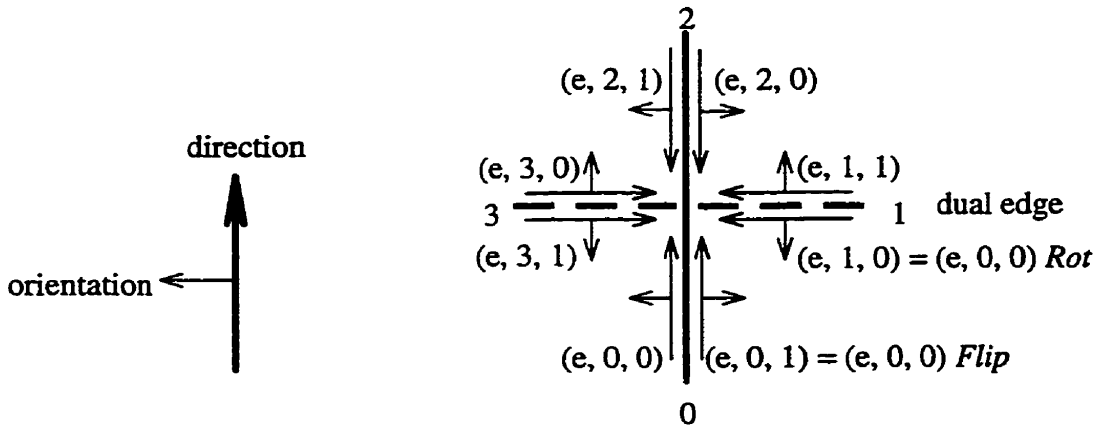
Define a subgraph $T = \{T_1, T_2, \ldots, T_m\} \subset D(S)$, $1 \leq m \leq 2n$, where each $T_i \subset T$ is a 2-simplex whose composing 0- and 1- simplices belong to S, and E, respectively. The only constraint on $T_i \subset T$ is the empty circumcircle criterion, that is, the circumcircle defined by the vertices of each 2-simplex in T does not contain any object in S in its interior. The subgraph $T \subset D(S)$ is called the Delaunay triangulation. Each $T_i \subset T$ is a Delaunay triangle.

3.5 The Data Structures of the Voronoi Diagram

Designing a data structure to represent the Voronoi diagram is the first concrete step in the construction of the diagram with a computer, in addition to understanding the nature and properties of the diagram. Following the discussion presented in the preceding section, the design of the data structure is centred on representing the dual Delaunay triangulation. The following two important requirements are imposed on the data structure: 1) It must be topological in the sense that only the identifiers, not the geometry, of an object class should be involved in the supporting operations for topological properties; and 2) The transition between an identifier of an object and its geometric definition should be natural and smooth. There could be many different data structures encoding a Delaunay triangulation, although two representations stand out. The first one is the well-known *quad-edge* data structure [Guibas and Stolfi 1985], and the other is the *triangular element* data structure [e.g. Gold 1976; Gold et al. 1977].

The quad-edge data structure is a computer implementation of the *edge algebra* developed by Guibas and Stolfi [1985] which captures all the topological properties of the subdivision of a surface. Each undirected edge of a non-oriented subdivision is composed of eight information items: four encode the orientation and direction of the edge, and the other four encode that of the dual edge (linking left and right faces). Figure 3.16 depicts the direction and orientation of an edge in (a) and a generic edge structure in (b). For a generic edge e, its record, identified as e, has four parts (quarters), e[r] with $r \in \{0, 1, 2, 3\}$, plus in each part one additional bit $f \in \{0, 1\}$. Therefore, the edge e can be referenced by the triplet (e, r, f), where r indicates the orientation, and f the direction. This triplet serves as a pointer to a "record-quarter" e[r] plus the bit telling us the edge e by a particular vertex, and a particular face. Each part e[r] of an edge record e can contain two fields, Data and Next. The Data field holds geometric and attribute information about the edge e in orientation r and at terminal f. The Next field contains a reference to an adjacent edge in a counterclockwise order. The algebraic operations upon quad-edges include *Flip*, *Onext*, and *Rot*. The *Flip* operation on an edge e returns the same unoriented edge with the opposite orientation and the same direction. The *Onext* operation on e returns the edge immediately following e in a

counterclockwise direction, with the same origin as *e*. The *Rot* operation on *e*, however, returns the dual of *e*, which is *e* being rotated 90° counterclockwise around the crossing point of dual edges. The quad-edge data structure contains no separate records for vertices or faces. A vertex is implicitly defined as a ring of edges and can be referred to by specifying any of its outgoing edges. On an orientable manifold, such as the Euclidean plane or the sphere, the Flip operation is not needed.



a) A directed, oriented edge b) A quad-edge e

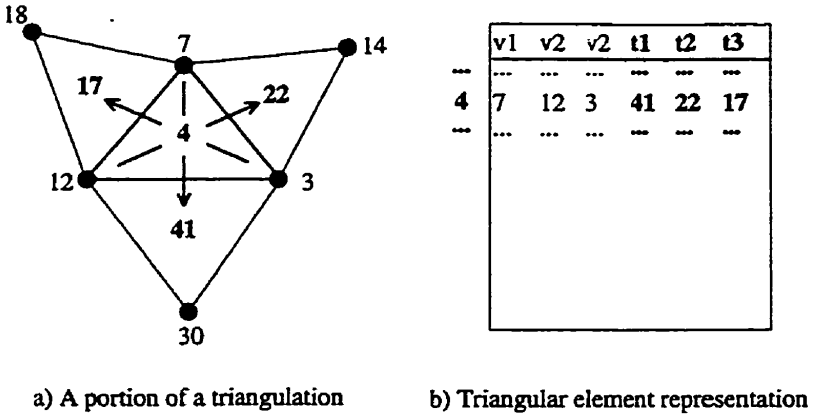
Figure 3.16 An illustration of the quad-edge data structure

The triangular element data structure is simple, and more intuitive. As the name suggests, the basic elements of the data structure are individual triangles. For each triangle T_i in a triangulation, a tuple of six ordered pointers is used:

$$T_i(v_1, v_2, v_2, t_1, t_2, t_3),$$

where v_1, v_2, and v_2 are three system-generated references for the vertices of the triangle in counterclockwise order; and t_1, t_2, and t_3 are three system-generated references for the adjacent triangles. The order of the adjacent triangle references is arranged such that for each t_k, $k \in \{1, 2, 3\}$, its referred triangle shares a triangle edge with triangle T_i, the shared triangle edge must be defined by v_{k-1} and v_{k+1}. In other words, the position of t_k

corresponds to the position of vertex v_k. Figure 3.17 shows a portion of a triangulation and its data structure. The arrows illustrate the positioned correspondence between vertex and adjacent triangle references.



a) A portion of a triangulation b) Triangular element representation

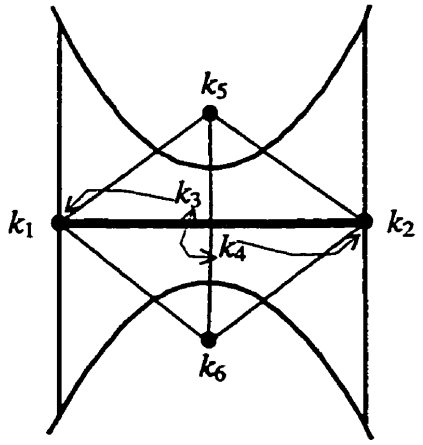
Figure 3.17 The triangular element data structure

Unlike the quad-edge data structure, the triangular-element data structure explicitly represents the identities of triangles and their vertices but not the triangle edges. A triangle edge is implicitly identified from the data structure. Both data structures are equivalent for a planar triangulated subdivision in the sense that topological properties involved in this subdivision are represented. The triangular element data structure is used to represent and store Voronoi diagrams in this thesis. This was chosen for the historical reason that the surface interpolation data model was first studied based on the triangular element data structure [e.g. Gold 1976, 1989], which has been transformed later into the fundamental data structures for a dynamic spatial data handling system [e.g. Gold 1991].

One of the most important extensions to the original triangular element data structure permits the Voronoi diagram to handle both points and line segments [Gold 1990]. This capacity is built into the associative topological object structure to represent and manage these two types of objects. As is shown in Figure 3.18a, the object structure specifies the relationship between a line segment and its endpoints. It has been mentioned before that a line segment has to be distinguished by its sides, it is therefore oriented. It is also directed if

the start and end points of the line segment can be identified. This is done by actually treating a line segment composed of two mutually associated and oppositely oriented lines. When a line segment is referred, it is necessary to indicate which side.

Let $k_i \in I$, $w_i \in I$, $1 \leq i < \infty$, and I is a finite, distinct set of positive integer numbers. The example of a general line segment in Figure 3.18a has two oriented lines k_3 and k_4. For the oriented line k_3, the endpoint k_1 is associated with it; and for k_4, the endpoint k_2 is associated (indicated by dashed arrows). Associated also with an oriented line is the other side of the line (dashed arrows relating k_3 and k_4). The ending point of the oriented line k_3, can be retrieved by asking for the starting point of the oriented line at the other side, which is k_4. The data structure representing this scheme is shown in Figure 3.18b. The distinction between a line segment and a point is made in the "Other_side" field. For a point object, the value of its "Other_side" field is null (i.e. 0), and the value of its "End_point" field will never be a null. For an oriented line, the "End_point" field is filled with the identifier of the starting point; and the "Other_side" field is the identifier of the associated oriented line on the other side.



(a) The associative object structure

OID	Endpoint	Other_side	...
...	...	...	...
k_1	w_1	null	...
k_2	w_2	null	...
k_3	k_1	k_4	...
k_4	k_2	k_3	...
k_5	w_3	null	...
k_6	w_4	null	...
...	...	...	...

(b) The object structure representation for the data in (a)

Figure 3.18 The associative object data structure

The subject of identifying orientation of geometric lines and planes has been recognized in computation geometry. One theoretic treatment [Stolfi 1987] named the study as "oriented projective geometry". The oriented projective geometry can be viewed as a marriage of classical projective geometry, which underlies the homogeneous coordinate representation, with an algebra of orientations, which is the ordinary algebra augmented by two-sided representation of objects.

3.6 The Construction of the Dynamic Voronoi Diagram

The computing of a Voronoi diagram can be carried out directly, or by constructing its dual, the Delaunay triangulation, since they are topologically equivalent. The construction through the Delaunay triangulation is especially advantageous if the data structures are oriented to representing the Delaunay triangulations. By associating triangles with some relevant geometric information, for example the coordinates of the corresponding Voronoi vertices, we are computing, simultaneously, the underlying Voronoi diagram.

There are typically four approaches to constructing a Delaunay triangulation. The algorithms of each approach are extensively studied in the computational geometry community. The *divide-and-conquer* [Drysdale and Lee 1978; Guibas and Stolfi 1985] approach recursively divides the set of objects into two equally smaller subsets, usually the left and the right halves. The Delaunay triangulation of each half is then recursively computed and finally merged with that of the other half into a bigger one, until the whole set is spanned with one Delaunay triangulation. The *plane-sweep* method [Fortune 1986, 1987] uses a horizontal line, called the sweepline, and moves it over the plane from bottom to top, halting at special points, called "event points". The Voronoi diagram is constructed along this line by maintaining a list of Voronoi regions and boundaries encountered by the sweepline, and a priority queue of events. The third approach is rather different in that it is based on the idea of "transforming geometrical problems into more easily understood and solved ones". This method [e.g. Brown 1979; Edelsbrunner 1986] transforms Voronoi diagrams in R^2 into convex hulls in R^3: the points are mapped, via a stereographical

projection, into points lying on a sphere. For this reason, the method is called "*lifting-up*" [Okabe et al. 1992] or "*higher dimensional embedding*" [Aurenhammer 1991]. The convex hull in higher dimensions is finally transformed inversely to the original plane. All these three methods can compute a Voronoi diagram with the optimal $O(n \log n)$ time bound, where n is the size of the object set. However, a serious drawback of these algorithms is the assumption that all objects are known before these algorithms are applied, which is unrealistic in a dynamic environment. The fourth approach is the *incremental method* which is the simplest to understand and implement. The incremental approach is usually attributed to Green and Sibson [1977] for their algorithmic description of the method, although Gold [1977] independently discovered this approach and used it to generate contour maps based on a partial ordering of the triangulated plane. The idea of this method is first to set up a Delaunay triangulated frame of the universal plane and then to modify the triangulation, based on the empty circumcircle test, as new points are inserted one at a time into the plane.

We are interested in the incremental approach to construct *dynamic* Voronoi diagrams. "Dynamic" means that the diagram is (locally) modifiable as objects are added or deleted. We chose this method for the following reasons: Firstly, most geographical applications experience dynamic processes involving spatially referenced objects, as identified in the first chapter of this thesis. Often a dynamic process occurs in an existing spatial configuration which is then interacted with and modified during the process. A new spatial configuration is derived as a result of the addition or deletion of objects, or the displacement of a previous configuration. The status of the set of spatial objects in the spatial configuration is therefore dynamic in nature. Secondly, the incremental approach is naturally akin to the temporality of a geographical database with which the history of the evolution of spatial objects can be tracked. This is sensible because the tracking process corresponds directly to a geographical process. Thirdly, taking the construction of the Voronoi diagram as a cartographic application, the incremental approach could be advantageous when used under some interactive environment. All cartographic data inputting and editing processes can be monitored and validated within a topologically meaningful framework, hence avoiding the tedious follow-up process of correcting

topological errors. Finally, the incremental approach permits the building of an object-oriented and dynamic spatial database which is concurrently accessible by multi-users and supports parallel processing of geographical problems. These arguments will be made clear as the thesis proceeds.

The Kinematic Construction Algorithm

The particular incremental method applied in this thesis is based on the *kinematic construction algorithm*, with which embedding objects are allowed to move along given continuous trajectories. The kinematic procedures to generate Voronoi diagrams of points and line segments have been practiced in the GIS community since the early 1990s [Gold 1991, 1992a]. They have been simultaneously investigated in the community of computational geometry [Roos 1991; Guibas et al. 1991], with the emphasis on the independent motions of a set of points in the plane. A recent publication [Gold et al. 1997] on fully dynamic and kinematic Voronoi diagrams and their application in GIS demonstrates a fruitful collaboration between the two communities.

Given a set $S = \{s_1, \ldots, s_n\}$ of points and line segments on the Euclidean plane R^2. We assume that an initial Voronoi diagram $V(S)$ exists. The kinematic algorithm dynamizes $V(S)$ by inserting new elements into, or deleting one from the set S. It also displaces point objects in S. The objective of the algorithm is to maintain a continuously updated Voronoi diagram $V(S)$ for the dynamic object set S. This is achieved by maintaining a current topological structure of $V(S)$, the Delaunay triangulation $D(S)$. The concept of how to detect potential change of $D(S)$ is different from other dynamizing, but non-kinematic techniques (compare, e.g., Boissonnat et al. [1992] and Devillers et al. [1992]).

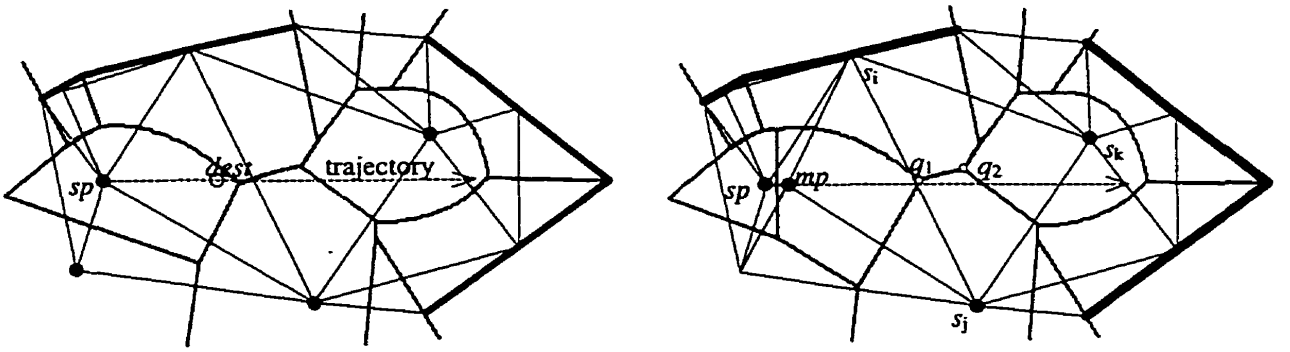
Nearest object search (nos). We devise a nearest object search function (nos) and use it extensively to assist in the construction of and query operations on a Voronoi diagram. Searching for a nearest object embedded in a Voronoi diagram is a specific case of a general *nearest-neighbour problem* [Knuth 1973]. By constructing a hierarchical Delaunay search structure with n dynamically moving points, using $O(n \log n)$ time and $O(n)$ storage,

Roos [1991] proved that each nearest-neighbour query can be affected in worst-case optimal $O(\log n)$ time. Even by a simple walking algorithm [Green and Sibson 1977; Gold 1977] over a triangulated structure, Guibas and Stolfi [1985] analyzed that an $O(n)$ worst-case time bound suffices for locating a point. Our nos function is based on the simple walking procedure and has been shown to be efficient in an interactive environment.

Inserting, deleting, moving a point

Destination, starting point, trajectory, and moving point: Before a new point s is inserted, its location vector $s \in R^2$ must be given which is the destination of s, denoted *dest*. Superimposing the destination s onto $V(S)$, the nearest object to s can be found from S through $D(S)$ with the nos function. To insert a point, we name the nearest object the starting point for modifying $D(S)$, denoted *sp*. A trajectory is a half-line in R^2, emitted from $sp \in S$ and passing through *dest*. A moving point, denoted *mp*, is created and inserted into S. The initial location of *mp* is the same as that of *sp* and is immediately moved away from *sp* along the trajectory. Gold [1991, 1992a] describes this operation as *object split*.

Consequently the topological structure, $D(S)$, has to be modified to $D(S \cup mp)$. The modification is in respect to the newly split Voronoi region $v(MP)$ which has similar neighbouring Voronoi regions as $v(sp)$ before *mp* is split, plus now $v(sp)$. Figure 3.19 illustrates the above concept where in (a) a destination for a new point is superimposed on a Voronoi diagram and the starting point *sp* is found. This determines a trajectory (the dashed and arrow line). The moving point *mp* is then split and displaced (exaggerated in the diagram) away from *sp*, together with the Voronoi diagram modified in (b). For the sake of clarity, the destination mark is removed from Figure 3.19 (b). Instead, two Voronoi vertices are labeled as q_1 and q_2. It can be verified that q_1, q_2 are the centres of the circumcircles defined by two triangles $\Delta mp\text{-}s_j\text{-}s_i$ and $\Delta s_i\text{-}s_j\text{-}s_k$, respectively. The reason for this labeling will soon be made clear.



(a) before splitting a point (b) after splitting a point

Figure 3.19 Concepts related to splitting a moving point in point insertion

After splitting, *mp* exists in *S* and moves to it's destination. It is known that *mp* is a Delaunay vertex shared by a number of adjacent triangles, determined by the number of Voronoi edges for *v*(*mp*). As *mp* moves, the shape of *v*(*mp*) will be reformed, accompanying the displacement of the Delaunay vertex designated by *mp*. Nevertheless, so long as the circular list of the adjacent triangles around *mp* does not change, the topological structure stays the same. This can be seen from Figure 3.20 where mp is moved along the trajectory some distance without changing the topological structure established in Figure 3.19 (b). We are, however, interested to know when the topological structure should be altered and how we can be informed of this possible change. By carefully studying Figure 3.20, one can observe that the Voronoi edge between q_1 and q_2 in Figure 3.19b becomes zero, and q_1 and q_2 coincide to a single Voronoi vertex, q, which is now intersected by four Voronoi edges. This indicates that Voronoi vertex q must be correspondent to four cocircular objects. They are, indeed, the four triangle vertices of the two adjacent triangles, $\Delta mp\text{-}s_j\text{-}s_i$ and $\Delta s_i\text{-}s_j\text{-}s_k$.

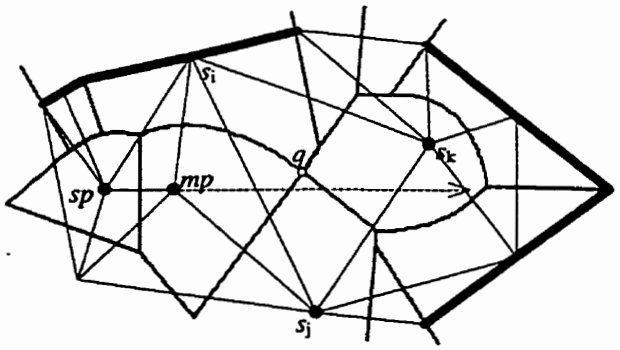


Figure 3.20 Moving *mp* without changing the topological structure

The phenomenon, that a degenerate Voronoi vertex becomes contracted with *mp*, signals that the topological structure might be modified. Roos [1991] calls this signal a *topological event*. There is yet another type of topological event which corresponds to the phenomenon, that *mp* is about to abandon a contraction with an existing degenerate Voronoi vertex. Gold [1991, 1992a] distinguishes these two topological events as moving-in, for the former type, and moving-out, for the latter one (Figure 3.21).

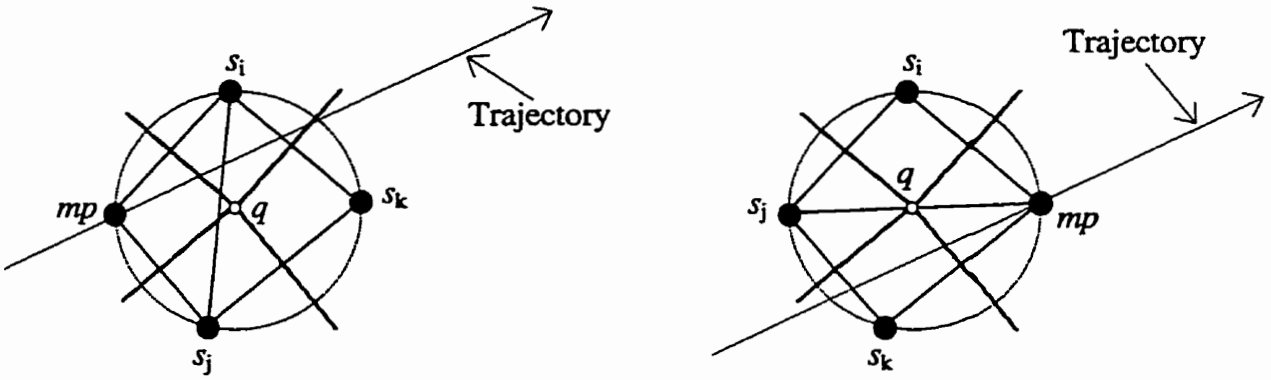


Figure 3.21 Two types of topological events

Intuition is sufficient to understand the two topological events. For both events, the moving point *mp* arrives at a critical point. It is either moving into (for Figure 3.21a) the circumcircle originally defined by triangle Δs_i-s_j-s_k, or moving out of (for Figure 3.21b) the

circumcircle defined by triangle Δs_i-s_j-s_k. The kinematics of the algorithm capture all topological events activated by the advance of the moving point. For each event, a decision has to be made on whether or not an action should be taken. The action, actually the well-known *swap* function [Green and Sibson 1977; Gold 1977; Guibas and Stolfi 1985], switches the triangle edge diagonal to the quadrilateral defining the circumcircle corresponding to the degenerate Voronoi vertex, q. A decision can be justified based on the known empty circle property that any topological conflict should be avoided. Figure 3.22 illustrates the result of swapping, with (a), (b) corresponding to (a) and (b) in Figure 3.21, respectively. It can be seen that after the swap, the contraction of *mp* with a degeneracy is released. The circumcircle in Figure 3.22 (a) should not exist, we keep it there to show the moving-in idea.

It should be noted that a swap is a local operation in that the only triangles which have to be adjusted are the two triangles in the quadrilateral and the four additional triangles sharing a boundary with the quadrilateral (Figure 3.23).

The process of detecting topological events and swapping triangles to resolve possible topological conflicts continues until *mp* arrives at its destination.

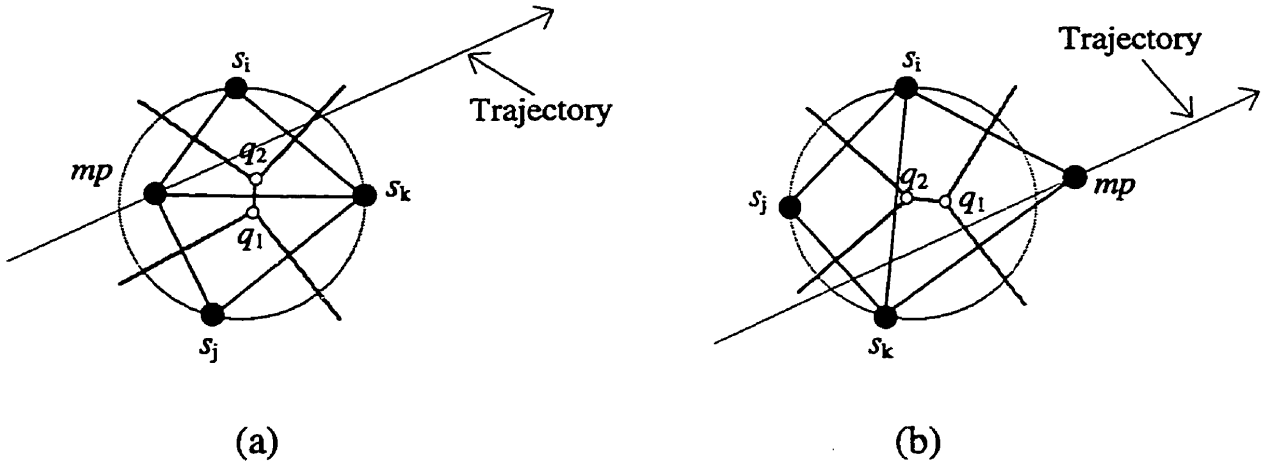


Figure 3.22 Two swaps corresponding to moving-in (a) and moving-out (b)

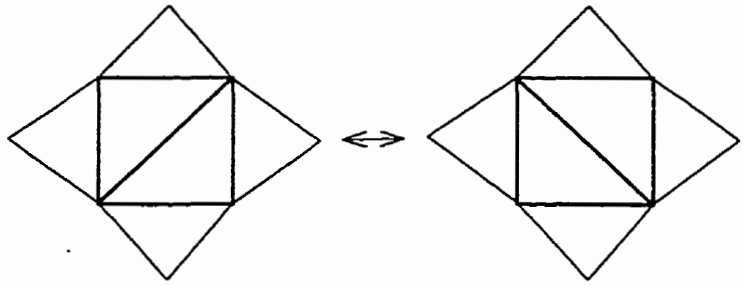


Figure 3.23 Triangles affected by a swap

Deleting a point $s_i \in S$ is a similar process to point insertion. In this case, the point to be deleted is already in S therefore there is no need to split a point. In addition, the destination is not given but is instead chosen as the location vector occupied by the nearest point in S. After this, the trajectory is known to be the half-line emitted from the original location of s_i and passes through the selected destination. The point s_i then becomes *mp* and is moved towards the destination. When *mp* arrives close to the destination point, with the topological structure around it similar to that of a point splitting, a *merge* [Gold 1991, 1992a] operation takes place. The merge operation reverses the split process.

Moving a point $s_i \in S$ to a designated location (the destination and the trajectory are known) is simpler because it does not involve splitting (as in point insertion) or merging (as in point deletion).

Inserting and deleting a line segment

A line segment can be connected and unconnected. Inserting an unconnected line segment into S is divided into two steps: inserting a point into S, which will be an endpoint of the line segment; then inserting the line segment from the endpoint. Similarly, inserting a line segment into S and connecting it to an object already in S also involve two steps: designating a point in S to be an endpoint of the line to be inserted; then inserting the line segment from the endpoint. In either case, the location vector of the destination has to be given, which will be the location of the other endpoint of the line segment. The trajectory is then determined to be emitted from the endpoint in S and passing through the destination.

Let $sp \in S$ be the designated endpoint of the line segment to be inserted. Similar to splitting a point, three new objects will be created and inserted into S. The three objects are: the other endpoint, denoted *mp*, meaning it will be the moving point; the left-side line, *ll*; and the right-side line, *rl*. Both side lines are collinear to the trajectory and are mutually referenced: *ll* is referenced to *sp*, thus making the new line segment connected to a point in S; *rl* is referenced to *mp*. The initial location of *mp* is the same as that of *sp*. Conceptually, three Voronoi regions, $v(mp)$, $v(ll)$, and $v(rl)$ are to be split from $V(S)$. This is achieved by modifying the topological structure, $D(S)$, to be $D(S \cup mp \cup ll \cup rl)$. Figure 3.24 illustrates the Voronoi diagram in Figure 3.19 (a) immediately after a new line segment is split from *sp*. The exaggerated displacement of *mp* demonstrates the modified topological structure so far.

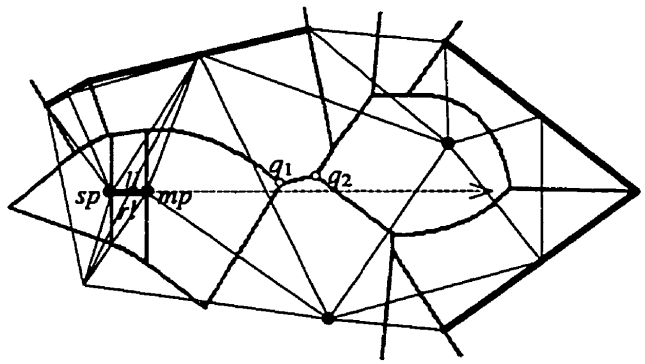


Figure 3.24 Splitting a line segment from an object in S

The moving point, *mp*, now moves toward the destination, dragging a line behind it. The topological structure around *mp* is modified to correspond to the topological events detected. It is worth noting that the behaviour of the moving-in topological event is the same as inserting a point (Figure 3.25). It is different, however, for the moving-out event. The distinction is that the trajectory is always tangent to, but never cuts, the circumcircle that *mp* is leaving (Figure 3.26). Otherwise the empty circumcircle condition would be violated.

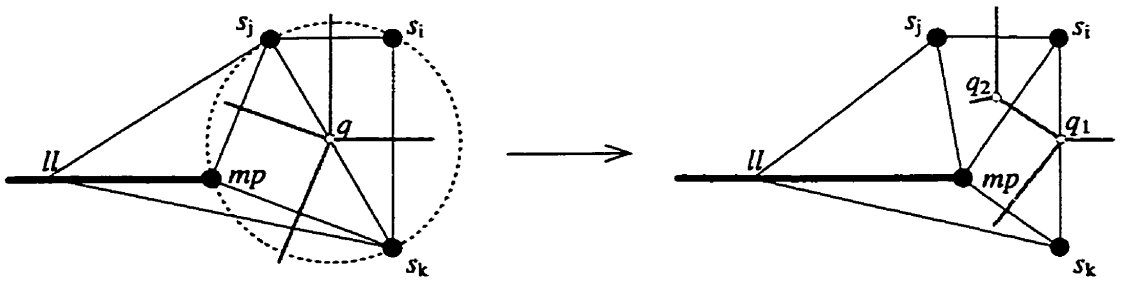


Figure 3.25 The topological structures before and after moving-in

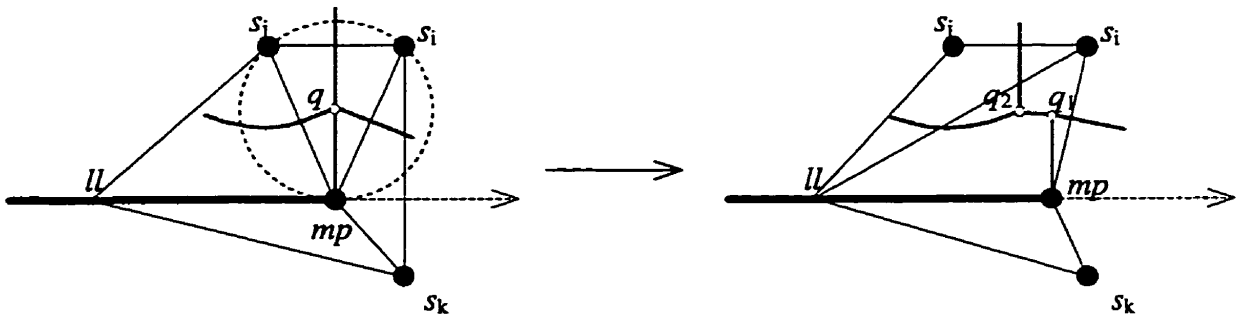


Figure 3.26 The topological structures before and after moving-out

Sometimes it is desirable to insert a line segment which has both endpoints connected to two objects in *S*. In this case the destination endpoint should also be designated in *S*. Similarly, the moving point and two oriented lines need to be inserted in *S*, with the starting endpoint connected. When *mp* nears the destination and there will be no more moving-in or moving-out events between the current location of *mp* and the destination, the merge operation takes place, which changes the reference point of *rl* from *mp* to the designated endpoint in *S*.

Deleting a line segment in *S* consequently deletes all unconnected components of the line segment; first, delete the interior of the line segment; second, delete any unconnected endpoints of the line segment. To delete the interior of a line segment, one endpoint of the line segment is designated as *sp* and the other endpoint as *tp*. The trajectory starts from *sp* and passes *tp*. A new point is then created and inserted into *S*. This point is designated as

mp, and is split from *sp*. In splitting *mp*, the reference point of the left-side line is changed from *sp* to *mp*, and the topological structure for both *sp* and *mp* is constructed. Figure 3.27 shows the result of splitting *mp* before starting to delete a line segment.

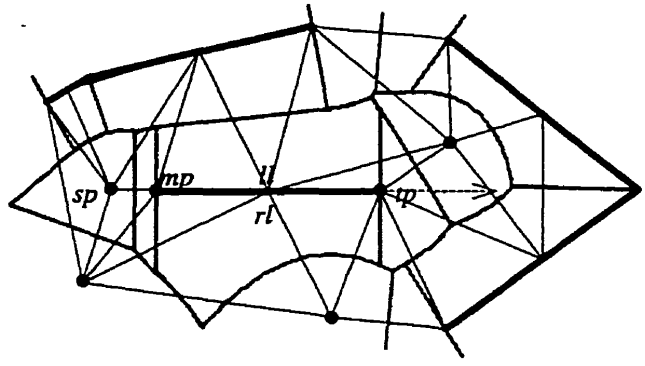
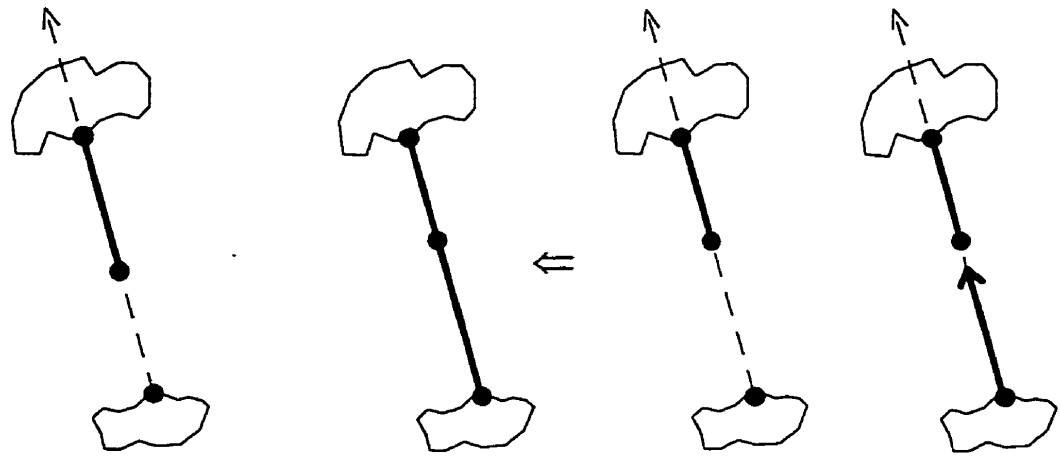


Figure 3.27 Splitting *mp* to delete a line segment

The topological events are then captured and the topological structure is modified as *mp* moves to *tp*, the destination. Analogous to deleting a point, when *mp* arrives at a location close to *tp* and there are no more topological events left, the merge operation finally adjusts the topological structure around *mp* and removes *mp*, *ll*, and *rl* from the set *S*.

With the basic inserting and deleting algorithms for handling line segments, it is straightforward to modify and extend them to suit other cartographic operations. For example, instead of completely deleting a line segment, the line can be shrunk to a point on the line segment. Furthermore, a line segment can be broken into two line segments, connected at the breaking point. Breaking a line segment may be done by first shrinking the line segment to the breaking point, and then inserting a new line segment in between the original endpoint and the breaking point. Figure 3.28 illustrates shrinking, and breaking operations on a line segment. Clearly, breaking a line segment is a compound operation.



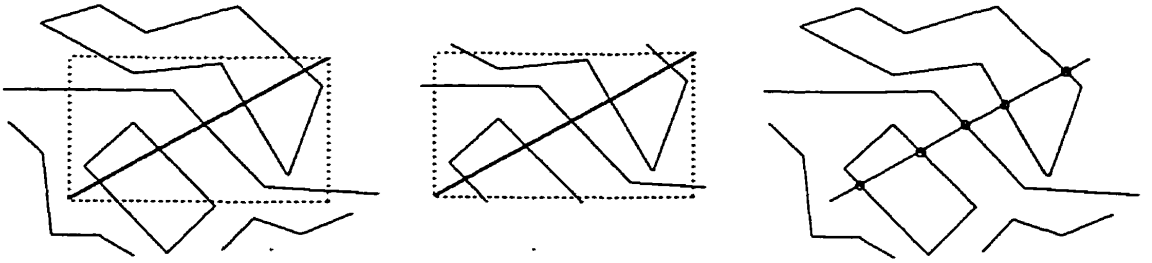
(a) Shrinking a line segment

(b) Breaking a line segment: 1) shrinking, 2) inserting

Figure 3.28 Shrinking and breaking a line segment

The dynamic resolution of the line intersection problem

Inserting a line segment into the object set S may cause a problem in that the new line segment intersects a number of existing objects already in S. Correctly finding intersections of a line segment with other line segments in S is not trivial if the objects in S are not supported by an efficient indexing structure. Even with some indexing structure, the usual approach will first involve a range search with respect to a minimum bounding rectangle (MBR) for the full range of the new line segment. The intersection test then has to be applied to each line segment falling inside the bounding rectangle. Consequently, intersecting nodes, together with the new line segments incurred, have to be inserted into the indexing structure (Figure 3.29). If a Voronoi diagram is constructed for the set S with a non-kinematic incremental method, the first two steps are still applicable when a new line is inserted. In this case, of course, the third step is not to modify the indexing structure, but to rigorously modify the Voronoi diagram.



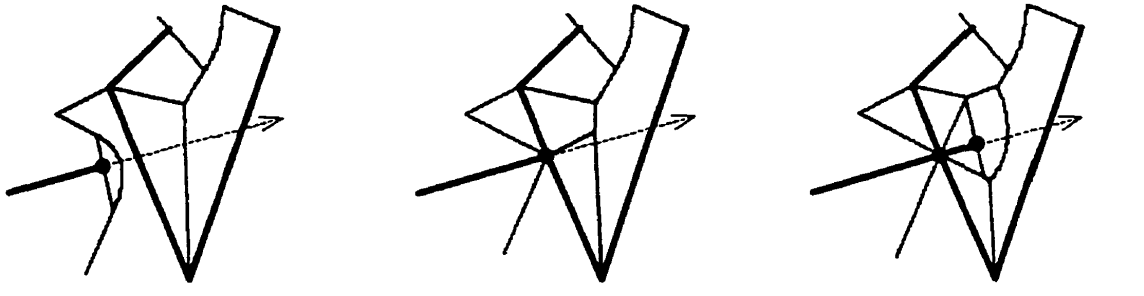
(a) Form a MBR for the new line segment

(b) Range search for all line segments in MBR

(c) Calculate intersections and update the geometric data structure

Figure 3.29 The usual approach to resolving the line intersection problem

The kinematic approach resolves this problem in an unusual and elegant way. The solution is topological rather than geometrical in that 1) a potential intersection is captured by detecting a topological event (no range search is needed); and 2) the sequence of intersections is known with regard to the direction of the trajectory. With the kinematic method (Figure 3.30), an intersection is detected dynamically as the moving point is proceeding to a given destination. A possible intersection is signaled by a topological event which indicates that the moving point is about to collide with an existing line segment (Figure 3.30a). This signal arises when the moving point is left with only two neighbours. If an intersection is about to and is intended to happen, the colliding line is broken at the intersection and a join node is made to connect the three line segments (Figure 3.30b). After this, a new line segment is split from the join node, and pulled along the trajectory by a moving point towards its final destination (Figure 3.30c).



(a) A collision is detected

(b) Break the line segment

(c) Pull the line after the break

Figure 3.30 The kinematic method of handling line intersections

The kinematic method of handling the problem of line intersection is readily applicable to deal with the collisions of a moving point with obstacles in a spatial configuration, and can be applied to advantage in robot navigation, motion planning, and GIS intelligent digitizing [Gold et al. 1996]. For example, Gold and Condal [1994] described a simulation model for boat navigation in a marine environment.

3.7 Preserving the History of Changes in Spatial Objects

An increasing demand on a spatial data handling system is that the history of changes in spatial objects needs to be preserved. The history of a spatial object records the time when the object was created and inserted into the space, when any changes have occurred on its geometric shape and placement, and if it was removed from the space. The complications of creating, transforming, and destroying spatial objects are discussed in Worboys [1995]. If organized and structured, the history of spatial objects in a system can be queried and retrieved, thus presenting the temporality of the system. Moreover, one can talk about a spatial object by referring to a specific time instance or period, implemented with an underlying temporal framework. Spatial objects referenced with both spatial and temporal dimensions are called *spatio-temporal (ST) objects* (Worboys [1995], pp. 303-314).

Recent years, research and development on spatio-temporal database management systems have been actively carried out in the Centre de recherche en geomatique at Université Laval, by Dr. Bédard and his team. Substantial results from a series of inter-related projects have been achieved. Bédard and van Chestein [1995] presented primary requirements in managing the temporality of geo-objects. Gagnon [1993] discussed notions and concepts involved in space and time. Proulx [1995] developed a query language and an interactive interface for a spatio-temporal GIS based on graphical notions representing topological and temporal relationships. Szarmes [1997] addressed issues of spatio-temporal databases, and proposed a framework for modelling the evolution of spatio-temporal database structures. The framework is based on a management of modifications occurred in conceptual schema

whose structures are formalized using Modul-R 2.01, a conceptual data modelling tool, developed in the Centre de recherche en geomatique at Universite Laval [Bédard et al. 1994].

Closely related to the incremental approach is the feasibility of implementing a dynamic system to handle ST objects. As can be seen from the processes of inserting, moving, shrinking, breaking, and deleting a primitive spatial object in an object set, book-keeping procedures can be devised to keep track of these processes. A meaningful handling of ST objects, of course, must incorporate additional mechanisms which, for example, assemble primitive temporal records into records corresponding to the temporality of identifiable complex geo-objects. This can be provisionally achieved by incorporating spatio-temporal modelling tools at the conceptual level, such as the ones developed by Dr. Bédard's group.

In this section, we discuss a log file structure which keeps one-dimensional, sequential temporality of primitive spatial objects, with respect to the construction functions discussed in the preceding section. It is therefore the temporality corresponding to *system transactions* [Langran 1992]. The idea of the log file structure was proposed originally by Gold [1994] as an approach to replay (forwards and backwards) a map history like playing a movie. To this purpose, the minimum requirement is to enable the reconstruction of the Voronoi diagram for a set of map objects, either from the beginning of the map history forward in time, or from the current state of a map backwards to previous times.

The implementation of the log file structure starts with symbolizing the dynamic Voronoi diagram construction operations. We shall use letters '*P*' and '*M*' to denote the inserted and moving points, respectively; '*L*', and '*S*' as the action of inserting and shrinking a line segment; and '*D*' as deleting an object for either a point or a line segment. Since breaking a line segment is a compound operation, it will not be recorded in the log file. A record of eight fields is used:

(T, O, OID1, x1, y1, OID2, x2, y2)

where the T field is for the transaction time, the O field for a character representing one of the construction operations, OID1 and OID2 are two system identifiers for two objects, and each is followed by a pair of x and y coordinates. The meanings of T, and O are not ambiguous. The interpretation for two triples of identifiers and coordinates is, however, dependent on a particular operation recorded in the O field. Their use of log file information for forward reconstruction is explained in Table 3.1.

O	OID1	x1, y1	OID2	x2, y2	Remark
'*P*'	not used	*dest*	not used	not used	split a new point from the nearest point
'*M*'	the object to be moved	of this object	not used	*dest*	the object identified by OID1 is moved to *dest*
'*L*'	the starting point *sp*	of *sp*	null if dest end not connected; of *tp* if dest end connected	*dest*	inserting a line segment between *sp* and *dest*. The dest endpoint is connected if OID2 is not null
'*S*'	of *sp* endpoint	of *sp*	of the left-side line *ll*	*dest*	the line segment, identified by *ll*, is shrunken from *sp* to *dest*
'*D*'	the object to be deleted	of *sp*	not used	*dest*	if the object to be deleted is a point, its nearest object is searched is *tp*; if it is a line segment, OID1 is for *ll*, and *sp* and *tp* are known from the object data structure. The two pairs of coordinates are used for backward play

Table 3.1 The use of log file information for forward reconstruction

When reconstructing a map from the current state backward to previous ones, the log file is consulted in reverse order, from the most current log record to the oldest one. The meaning of the operation has to be reversed accordingly. The objective is to obtain necessary reversing construction operands such as *sp*, *mp*, *dest*, the trajectory. For example, if the log operation is '*P*', to insert a point, the reverse operation should be '*D*', to delete one. In this case, the first pair of coordinates are used to search for the point, *sp*, occupying that location, and at the same time, the nearest point, *tp*, to *sp* is found. After this the *dest*, and the trajectory can be known, and reverse construction can start.

Besides reconstruction, the log file also provides limited possibilities for spatio-temporal queries. The log file structure is indexed in the chronological order of a temporal space, *T*,

mapped to an underlying numerical domain. It is possible, therefore, to answer queries of the following types:

$$Q(TI, SI) \Rightarrow S_1 \subset S, \tag{3.1}$$

$$Q(TI, \infty) \Rightarrow S_1 \subset S \tag{3.2}$$

where Q is a query, $TI = [t_1, t_2]$, $t_1, t_2 \in T$, $SI \subset R^2$ are temporal and spatial intervals, respectively, and the symbol "$\Rightarrow$" means to return the result of a query. The query type (3.2) implies that the spatial interval is not specified; it can be the whole plane. With the primitive log file structure, it is difficult to answer queries of the type:

$$Q(S_1 \subset S) \Rightarrow TI. \tag{3.3}$$

Hence the log file structure favours queries whose answer is a spatial subset, and not a temporal interval. Implementation for the first two types of spatio-temporal queries is straightforward. Preserving temporal aspects of complex spatial objects corresponds to dynamic events that create and destroy these objects. Capturing the dynamic events relies on the search facilities to detect the effect of primitive construction operations. Additional data structures are needed to record events of complex objects and to support complex spatio-temporal queries.

3.8 GIS Operations with the Dynamic Voronoi Diagram

This section discusses how typical GIS functions can be realized with the dynamic Voronoi diagram for points and line segments. The discussion first concentrates on the graph-theoretic properties of space tessellations and the procedures for performing network analysis over planar map objects structured with the Voronoi diagram. Spatial searches, especially the range search procedure, are then discussed. Both types of GIS function are based on a common feature of traversing the dual topological structure, the Delaunay triangulation. In spatial analysis functions, we outline general algorithms for polygon

shading, buffer zoning, and polygon overlay, highlighting the advantages of the underlying Voronoi data model. Finally we demonstrate the utility of Voronoi diagrams applied to digital terrain modelling. It is noted that although the general ideas of spatial searches, network and spatial analysis have been published in the literature, the application of these ideas to the design of workable algorithms based on the Voronoi diagrams of points and line segments is firstly reported in this thesis.

3.8.1 Network Analysis

Many GIS applications work on large data sets depicted as points and line segments on planar maps. These map objects (points and line segments) may represent highway networks, airline routes, or circuits in a VLSI design. Knowledge about interconnections between objects helps to answer questions such as "how to get from location A to location B in the fastest or cheapest way" and "Is location A connected to location B". Answering these problems with a computer depends largely on how information about the objects and their connections are modelled, and on effective algorithms based on the model.

In this section, we view a set of points and line segments as an undirected, weighted planar graph, $G(V, E)$, a collection of vertices V (points) and edges E (line segments). It is the vertices and edges that are the essence of the graph and the algorithms over it. A path from vertex x to y in G is a list of vertices in which successive vertices are connected by edges in G. A simple path is a path in which no vertex is repeated. A graph with no cycles is a tree. A spanning tree of a graph is a subgraph that contains all the vertices but only enough of the edges to form a tree. A graph is undirected if all edges can be followed in both directions. Conversely, a directed graph has "one-way" edges only: one travels from x to y but not from y to x. In weighted graphs values are assigned to each edge that represent distances or costs. Figure 3.31 is an example of an undirected weighted graph which will be used to illustrate our graph algorithms. In Figure 3.31, integers are identifiers of vertices while floating-point values are weights representing the distance of each edge. Figure 3.32 illustrates the Voronoi diagram (gray lines) and the dual Delaunay triangulation (thin lines) used to construct the graph depiction (dark lines) shown in Figure 3.31.

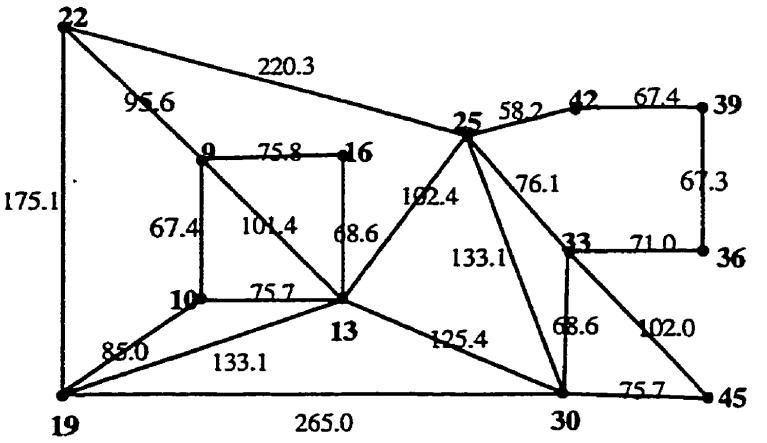


Figure 3.31 An undirected weighted graph

The Voronoi diagram is a connected graph for the whole space. Its dual, the Delaunay triangulation, is more interesting because it has map objects as vertices and its edges are all line segments. The simple geometric structure of the Delaunay triangulation makes the traversal of space an easy job. Similar to a linked list or a tree structure that must have a pointer directed to the starting address of the structure in the computer, the traversal of the Delaunay triangulation starts with a known triangle. No matter where the starting triangle is, by repeatedly walking adjacent triangles of a known triangle in a consistent scheme, the triangulation can be completely visited.

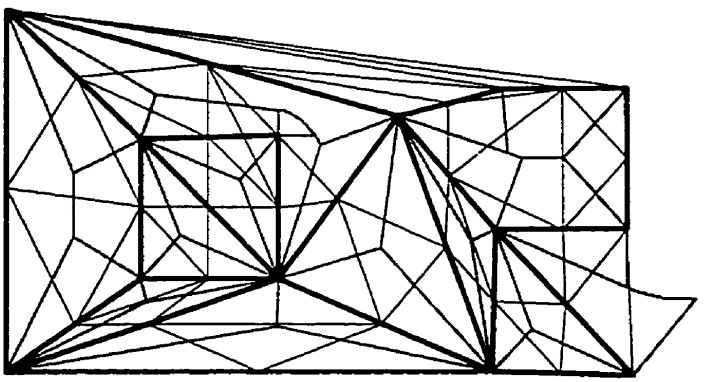


Figure 3.32 The Voronoi diagram and its dual Delaunay triagulation for the objects depicted in Figure 3.31

Given a graph of objects, there exist two classical graph-traversal algorithms which visit every vertex and edge in the graph. They are the *depth-first search* and the *breadth-first search*. The technique of examining every piece of the graph lays the ground for solutions to other graph-theoretic problems. In this section, we describe four classical graph algorithms using the Voronoi data structure developed in this chapter. We first present general strategies and subordinate data structures [Sedgewick 1992] to be used repeatedly. The applications of the general strategies lead to the algorithms solving the depth-first search, the breadth-first search, the minimum spanning tree, and the shortest-path problems. Further details about these strategies, subordinate data structures, and applications to other network analysis algorithms can be found in Sedgewick [1992].

Obtaining Incident Edges of Vertices

In graphic representations and algorithm designs, adjacent vertices, connected by edges, must be known. In the following algorithms, we assume that a function, called nbrs(*obj*), has been implemented over the Voronoi data structure. The nbrs(*obj*) function collects the neighbours of an object, named *obj*. Based on the definition of the Voronoi diagram, an neighbour is an object whose Voronoi tile shares a Voronoi edge with that of the concerned object. An array of integers *adj* is used to store the identifiers of neighbouring objects returned by the nbrs function. Note that in a Voronoi diagram, neighbours are not necessarily incident line segments to the given vertex (Figure 3.33). However, this can be easily screened by determining if one end of a neighbouring line segment shares the same node as the given vertex. In the *adj* array, the first neighbouring object can start from any of the neighbours, the rest of the neighbours are listed in anti-clockwise order. The searched neighbouring object list is terminated by the number 0. We use this tactic to determine the termination of loops in the algorithm.

(a) Neighbours of v are incident edges

(b) Not all neighbours of v are incident edges

Figure 3.33 The incidence of edges to a vertex

The General Strategy: Division of Vertices into Untouched, Fringe, and Tree

Several graph traversal algorithms use the same strategy of processing a graph. Prior to beginning an algorithm, all vertices are *untouched*. The algorithm [Sedgewick 1992] starts by visiting one vertex in the graph and putting it into a priority queue called a *fringe*. It then repeatedly removes a vertex from the fringe and inserts it into a *traversal tree* of vertices already processed. Adjacent vertices to the vertex just removed from the fringe are collected. Any untouched vertices in the graph are visited and put into the fringe. The algorithm finishes when the fringe is empty.

The above strategy is expressed as a general *priority-first* procedure called visit (Procedure 3.1, with the *C* language style). For the sake of simplicity, static arrays are used in the following procedures. The priority queue, *PQ*, is defined as an object class to be described. In the *visit* procedure, an instance of *PQ* is instantiated and named the *fringe*. An array *val* is used to record the order in which vertices are visited. The array is initialized to the value *unseen* to indicate that no vertex has been visited. The goal is to visit systematically all vertices of the graph, setting elements of the *val* for the *id*th vertex visited to *id*. Two additional functions are assumed: isline(*object*) which returns TRUE if an *object* is a line segment, and matchend(*vertex*, *line*) which returns the OID of the other end if one

endpoint of the line matches the *vertex*, or 0 otherwise. After calling *visit*, the traversal tree is established by collecting indices of the array *val* for values from 1 to *V*.

```
void visit(long v)      //general procedure
{  PQ fringe[maxV];
   long adj[MaxNumofNbrs];
   long adjv;
   int i;

   id = 0;
   fringe.insert(v);              //insert a vertex into the fringe
   while ( !fringe.empty() )
   {  v = fringe.remove();        //remove a vertex from the fringe
      val[v] = ++id;              //id-th vertex visited
      nbrs(v, adj);               //collect neighbours of v from the Voronoi diagram,
                                  //terminate the searched object list by 0
      i = -1;
      while ( adj[++i] && isline[adj[i] )      //for each adjacent line segment
         if ( (adjv = matchend(v, adj[i])) ! = 0 )
            if ( val[adjv] = = unseen )
            {  val[adjv] = -1; //mark the node in the fringe
               fringe.insert(adjv);
            } /*if val*/
   } //while (!fringe.empty())
}
```

Procedure 3.1 The general priority-first visit procedure

The Fringe: A Priority Queue

As we have just seen, a fundamental graph-search method is based on the step "move one vertex (call it *v*) from the fringe to the tree, then insert into the fringe any untouched vertices adjacent to *v*". One can think of the fringe as a record of nodes. Each node contains an integer field to hold the system-defined object identifier referring to a vertex of the graph (other data types may be possible, depending on applications), and a numerical field

to store the priority value of the vertex. We want the fringe structure to possess the following properties:

- Nodes can be processed in the order of their priority values, but the list is not necessarily in fully-sorted order and new nodes can be inserted dynamically;
- The first node in the list has the largest priority;
- Once the node with the largest priority is removed from the list, the node with the next largest priority should be ready;
- As new nodes are inserted, the node to be removed next should be updated accordingly, by comparing the priority values of all new nodes.

Based on the above specifications, the priority queue should therefore support insert and remove operations. Also, it should be able to perform swapping operations in order to prepare the queue with the largest priority. This can be implemented with a particular data structure called a *heap*.

The heap may be maintained as an array, a, of records such that each record is guaranteed to have a larger priority than the records at two other specific positions. In turn, each of those records must have a larger priority than two further records, and so on. The array satisfying the heap condition is a complete binary tree with the record of largest priority in the root which is the first position in the array. The heap implementation of the priority queue ensures that all operations in the priority queue can be done in logarithmic time. This property is important because swapping records in the array occurs at most of the priority queue operations. The code for the priority queue class is listed in Procedure 3.2.

```
struct node { long id; float w; };
class PQ
{  private: node _huge* a; int n;
   public:
      PQ(int max) { a = new node[max]; n = 0; }
      ~PQ() { delete a; }
      void upheap(int k)
      {  node v;
         v = a[k]; a[0].w = costMax;
         while (a[k/2].w <= v.w)
         {  a[k] = a[k/2]; k = k/2;  }
         a[k] = v; }
      void downheap(int k)
      {  int j; node v;
         v = a[k];
         while (k <= n/2)
         {  j = k+k; if (j < n && a[j].w < a[j+1].w) j++;
            if (v.w >= a[j].w) break;
            a[k] = a[j]; k = j; }
         a[k] = v; }
      void insert(node v)
        { a[++n] = v; upheap(n); }     //maintain heap properity with w value
      long remove()
      {  node v = a[1];                //first on the queue
         a[1] = a[n--];                //put the last on the queue to the first
         downheap(1);                  //maintain heap property
         return v.id; }
      long update(node v)
      {  int j, onfringe, changed;
         onfringe = changed = FALSE;
         for (j=1; j<=n; ++j)
            if (v.id == a[j].id)
            {  onfringe = TRUE;
               if (v.w > a[j].w)
               { a[j] = v; upheap(j); changed = TRUE; }
               break; }
         if (onfringe==FALSE) insert(v);
         if (onfringe == FALSE || changed == TRUE) return 1;
         else return 0; }
      long empty() { if (n == 0) return 1; else return 0; }
};
```

Procedure 3.2 The priority query class

When a new record is inserted, the number of records n in the heap must be increased by one. The new record is put into $a[n]$, but this may violate the heap property if the new record has a priority greater than its parent. This condition can be fixed by the upheap operation which repeatedly exchanges the new record with its parent until the heap condition is satisfied. In reverse, the remove operation takes the first record $a[1]$. Since the heap will be one record smaller, it is necessary to decrease n by one after moving the last record to fill the first position. The downheap operation is performed to correct the heap condition. All the basic operations -- insert, remove, upheap, and downheap require fewer than 2 log n comparisons when performed on a heap of n records. The empty method is a constant operation, it returns 1 if $n = 0$ and 0 otherwise. The update method will be explained when we discuss the minimum spanning tree algorithm.

In the following subsections, we explain how to set priority values for each record in the fringe and how to adapt the general visit procedure to different graph algorithms.

Depth-First Search (DFS)

The idea of the *depth-first search* is that it lets the priority of adjacent vertices reverse the sequence in which they are encountered. That is, the vertex last visited has a higher priority. Replacing the data type for the fringe and choosing the proper priority for each fringe node, the general visit procedure is modified for the DFS (Procedure 3.3)

```
PQ fringe;
void visit(node v)      //DFS with PQ
{  long adj[MaxNumofNbrs];
   node adjv;
   int i;

   v.w = 0;
   fringe.insert(v);              //insert a vertex into the fringe
   while ( !fringe.empty() )
   {  v = fringe.remove();        //remove a vertex from the fringe
      val[v.id] = ++id;                  //id-th vertex visited
      nbrs(v.id, adj);            //collect neighbours of v from the Voronoi diagram
                                  //terminate the searched object list by 0
      i = -1;
      while ( adj[++i] && isline[adj[i] )       //for each adjacent line segment
         if ( (adjv.id = matchend(v.id, adj[i])) ! = 0 )
            if ( val[adjv.id] = = unseen )
            {  adjv.w = id + i +1;      //DFS, the object visited later has higher priority
               fringe.insert(adjv);
               val[adjv.id] = -1;       //flag it on the fringe
            }
   } // while ( !fringe.empty() )
}
```

Procedure 3.3 The modified visit procedure for a depth-first search

In this procedure, the sentinel constant *unseen* is set to a large negative value. Running this procedure with the graph given in Figure 3.31, the traversal tree, starting from vertex 22 collected from the indices of array *val* by their increasing order of positive values is:

DFS Tree: 22, 9, 16, 13, 30, 33, 36, 39, 42, 45, 10, 19, 25

Breadth-First Search (BFS)

The algorithm for the *breadth-first search* traversal is almost the same as that for the DFS, except for a different choice of priority values for the fringe vertices. In the BFS traversal, it is the vertex put in the fringe earlier that will be removed first. Therefore, the adjv.w for

each of the adjacent vertices is assigned by a decreasing value, such that the one encountered earlier has a larger priority. In the general visit procedure, we modify the line

```
adjv.w = id + i +1;                    //DFS, the object visited later has higher priority
```

to become

```
adjv.w = maxnode - (id + i +1);        //BFS, the object visited later has lower priority
```

where maxnode is a positive constant specifying the maximum number of vertices in the graph. Running this modified procedure with the same graph, starting also with vertex 22, we have the tree traversal as:

BFS Tree: 22, 25, 19, 9, 13, 30, 33, 42, 10, 16, 45, 36, 39

Minimum Spanning Tree (MST)

A *minimum spanning tree* (*MST*) of a weighted graph is a collection of edges connecting all the vertices such that the sum of the weights of the edges is at least as small as the sum of the weights of any other collection of edges connecting all the vertices. The MST of a graph may not be unique, but it must be computed based on the following general property:

> Given any division of the vertices of a graph into two sets, the minimum spanning tree contains the shortest of the edges connecting a vertex in one of the sets to a vertex in the other set.

Based on this general property, we can build the MST by starting with any vertex and always taking next vertex which is the "closest" to the vertices already taken. In other words, we find the edge of lowest weight among those edges that connect vertices already on the tree to vertices not yet on the tree, then insert that edge and the vertex it leads to into the tree.

It turns out that the general visit procedure can be adapted to the construction of the MST with some modifications. For computing the MST, the priority of each vertex on the fringe should be the length (or cost) of the shortest (or cheapest) edge connecting it to the tree. For the same graph in Figure 3.31, if we use the Euclidean distance between a vertex just taken from the fringe and its adjacent vertices as the priority value for each adjacent vertex, we will end up with an *Euclidean Minimum Spanning Tree* (*EMST*).

The modification of the visit procedure for the MST is explained here. We use an array *dad* to contain the index of the parent for each vertex in the tree (0 if it is the root). A function *dist*($vertex_1$, $vertex_2$) is assumed to calculate the distance between $vertex_1$ and $vertex_2$. For each adjacent vertex, the update operation is performed on the fringe which ensures that the given vertex appears on the fringe with at least the given priority: if the vertex is not on the fringe, it is inserted, and if the vertex is there but the most recent one has a larger priority value, then the priority value is replaced. This must be followed by the upheap operation to correct the heap condition. If any change is made (either insertion or priority change), then the method update returns a nonzero value. This allows the visit procedure to keep the arrays *val* and *dad* current. To flag a vertex in the fringe, set its *val* equal to the negative distance value; to flag a vertex in the tree, set its *val* to the positive distance value. The priority value of each vertex is assigned the negative distance value to correspond to the internal operation on the fringe. A smaller negative distance of an edge will have a larger priority. Procedure 3.4 is the modified visit procedure for computing the EMST. It should be noted that the update method is not an efficient operation, since it may loop through all elements in the priority queue. The method can be improved at the expenses of maintaining a supplementary data structure. We will not however elaborate the improvement.

```
PQ fringe;
void visit(node v)      //MST with PQ
{  long adj[MaxNumofNbrs];
   node adjv;
   int i;

   v.w = unseen;
   if ( fringe.update(v) )              //insert a vertex into the fringe or change priority
      dad[v.id] = 0;                    //for the root
   while ( !fringe.empty() )
   {  v.id = fringe.remove();           //remove a vertex from the fringe
      val[v.id] = -val[v.id];
      if ( val[v.id] == unseen )        //true if v is the start vertex
        val[v.id] = 0;
      nbrs(v.id, adj);          //collect neighbours of v from the Voronoi diagram
                                //terminate the searched object list by 0
      i = -1;
      while ( adj[++i] && isline[adj[i] )       //for each adjacent line segment
         if ( (adjv.id = matchend(v.id, adj[i])) != 0)
            if ( val[adjv.id] < 0 )             //unseen or on the fringe
            {  adjv.w = -dist(v.id, adjv.id);   //EMST
               if ( fringe.update(adjv) )
               {  val[adjv.id] = adjv.w;        //flag it on the fringe
                  dad[adjv.id] = v.id;
               }
            }
   }  // while ( !fringe.empty() )
}
```

Procedure 3.4 The modified visit procedure for a EMST

Running this procedure for the graph in Figure 3.31, starting again with vertex 22, produces the following results:

MST:	id	22	9	10	13	16	19	25	42	39	36	33	30	45
	dad[id]	0	22	9	10	13	10	13	25	42	39	36	33	30

Shortest Path (SP)

The *shortest path* (*SP*) algorithm finds the path in a weighted graph that connects two given vertices x and y with the property: the sum of the weights of all edges of the path is

minimized over all such paths. It is easy to prove by induction that a breadth-first search starting at x will first visit all vertices that can be reached from x with one edge, then all vertices that can be reached from x with two edges, etc., visiting all vertices that can be reached with k edges before encountering any that requires $k + 1$ edges. Thus, when y is first encountered, the shortest path from x has been found because no shorter path reached y. In general, the path from x to y could touch all the vertices, so we usually consider the problem of finding the shortest path connecting a given vertex x with all of the other vertices in the graph.

The priority-first search solution to this problem is virtually identical to that for the MST, except that, for each vertex v taken from the fringe, instead of inserting the adjacent vertex closest to the tree into the fringe, one inserts the vertex which is the closest to v. To find which fringe vertex is closest to v, we use the *val* array for each tree vertex v. *Val*[v] is the distance from that vertex to its adjacent vertex *adjv*, using the shortest path. When an *adjv* is inserted into the fringe, we update the fringe using a similar approach to the one used for the MST. Compared to the visit procedure for the MST, the priority value of each adjacent vertex *adjv* of v becomes

```
adjv.w = - ( val[v.id] + dist(v.id, adjv.id) ); //Shortest Path
```

The rest of the code is the same.

Running the modified procedure using the graph in Figure 3.31, starting from vertex 22, we have the following results:

SP: id	22	9	10	16	19	13	25	42	33	30	39	36	45
dad[id]	0	22	9	9	22	9	22	25	25	13	42	33	30
val[id]	0	96	163	171	175	197	220	279	296	322	346	367	398

Values in *val* have been rounded for simplicity. Based on this result, the shortest path from vertex 22 to any other vertex can be found by tracing backwards the *dad* array. For

example, to find the shortest path from 22 to 33, we start from the vertex *id* 33 and find its *dad* 25, by the entry of 25 for the *dad*, we have 22. Therefore the desired shortest path is 22 - 25 - 33, with a minimum distance value of 296.

3.8.2 Spatial Searches

Spatial searches are frequently required in GIS, because of the need for spatial queries. Of the two types of spatial queries, point and range queries, the range query presents more stringent demands for efficient performance. A typical application of a range query displays all objects falling in a specified window, which may be regular, irregular, or degenerate to a line segment. In this section, we present, for the first time, a range search algorithm which utilizes the BFS or DFS strategy. The spatial data is constructed with the Voronoi diagram.

It turns out that a range search over the Voronoi diagram can be designed using the same scheme as the one for BFS or DFS. The difference for a range search is that the nodes in the fringe are triangles instead of map objects. This is because objects in a range may not all be connected but the triangles are. Therefore traversing through all the triangles overlapping the range and performing an inrange test on each vertex of a triangle visited ensures that all objects will be found correctly. The algorithm starts with one triangle overlapping the given range and inserts it into the fringe. It then repeatedly pops a triangle out from the fringe. The three vertices of the triangle are tested for inclusion or clipping the range. Included or clipped objects are reported and marked as visited. Adjacent triangles of the one being examined are retrieved and the ones overlapping the range and untouched earlier are inserted into the fringe. The algorithm finishes when the fringe is empty. A range_search procedure for a rectangular range, utilizing BFS, is included in Procedure 3.5, which is modified from Procedure 3.1. The existence of procedures get_adjtri, get_vertobj, overlap, mark_visited, and inclusion_clip are assumed.

```
PQ fringe;
int range_search(long seed, RRect rg, long* obj) //returns the number of objects found
{node v;
 long adjtri[3], mapobj[3];
 int i,j,k;
 int id=0;
 long count = -1;

 //the seed is the given triangle overlapping the range rg.
 v.w = 0;
 v.id = seed;
 fringe.insert(v);                    //insert the seed v
 while (!fringe.empty())
 {      v.id = id = fringe.remove();  //remove a node from the fringe, name it as v
        get_adjtri(v.id, adjtri);     //find adjacent triangles of v
        j = 0;
        for (i = 0; i < 3; i++)       //for each adjacent triangle of v
        {       v.id = adjtri[i];
                get_vertobj(v.id, mapobj);
                if (untouched(2, v.id))
                {       for (k = 0; k < 3; k++)
                        {       if (inclusion_clip(mapobj[k]) && untouched(1,mapobj[k])
                                        obj[++count] = mapobj[k];
                                mark_visited(1, mapobj[k]);
                        }
                        if (overlap(rg, v.id))
                        {       v.w = maxnode - (id + ++j);  //weight for the edge v-adjv
                                fringe.insert(v);
                                mark_visited(2, v.id);
                        }
                } //if untouched
        } //for each adjtri
 } //while !empty
 return count;
}
```

Procedure 3.5 A range search based on BFS

3.8.3 Spatial Analysis

Spatial analysis, as the term implies, analyzes a space. With objects embedded in space captured and modeled in a representation framework, this means obtaining qualitative and

quantitative properties of spatial objects based on the representation structures. There are numerous spatial analytic algorithms to calculate, derive, and collect various spatial properties. In this section, we give three examples of these algorithms based on the Voronoi data model and data structures. These ideas have been mentioned on different occasions, e.g. Gold [1994], Gold et al. [1997]. The following presentation is a more detailed description of these algorithms.

Identifying a Polygon (Polygon Shading)

A polygon is defined as the area enclosed by a closed loop of line segments. In most spatial data models, polygons are identified by a conceptual representation, but are not explicitly represented by a geometric data structure, as with the dynamic Voronoi data model. Visualizing a polygon in a spatial setting, or calculating its area, therefore, rely on a process to specify the extent of its geometric shape. This process is also referred to as *polygon shading* or *flood fill* with a raster-based system, and as *assembling polygonal chains* with a vector-based one. The algorithm based on the Voronoi diagram takes advantage of its integrated view of a polygonal space. As is illustrated in Figure 3.34, the interior of a polygon object is composed of Voronoi regions that are bounded by a closed loop. Each Voronoi region is associated with an oriented line segment. It is therefore natural to start with shading a known Voronoi region inside the polygon and expanding to shade adjacent Voronoi regions of each shaded one. The systematic way of doing this uses the triangular structure and is similar to the range search algorithm for traversing triangles. However, there is a difference in detecting if an adjacent triangle falls outside of the polygon. In range search, we used an overlapping test since the range is a regular shape and the range boundary is not part of the Voronoi structure. In polygon shading, the boundary of an arbitrary polygon is part of the Voronoi structure. Therefore, it is only necessary to test if two vertices of an adjacent triangle are boundary line segments but are on the other side. If the test is positive, the adjacent triangle will not be inserted into the fringe. During this flooding process, an attribute value can be assigned to each half-line segment enclosing the polygon, if desired. This method is capable to identify polygons with holes. Adjacent polygons can also be identified through the same method.

It is evident in this example that the Voronoi data model, although implemented with a vector-based data structure, conveniently uses algorithms based on the field-based view and raster data structures. This observation is also true for the polygon overlay algorithm to be described soon.

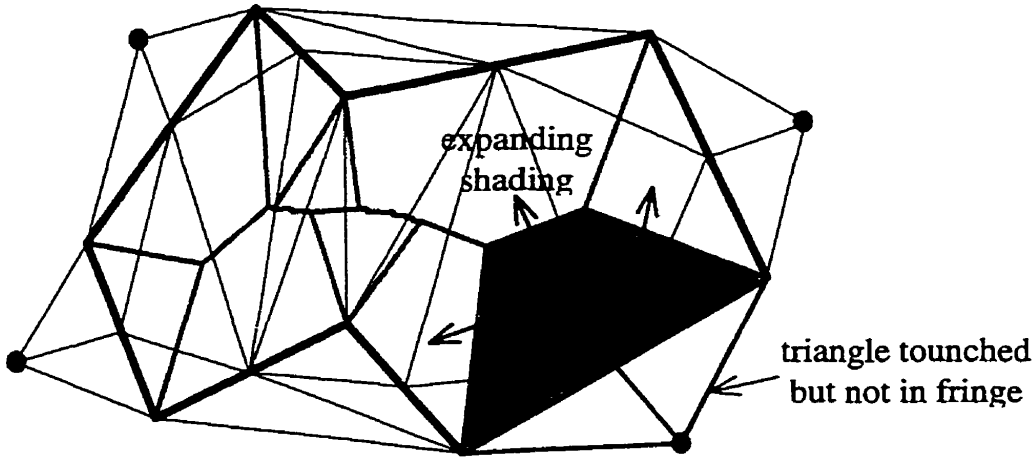


Figure 3.34 Polygon shading with the Voronoi data model

Buffer Zoning

Buffer zoning is the process of delineating areas within a given range of a set of objects. In a traditional vector GIS, determining buffer boundaries requires extensive calculations of the intersections of line segments and circular arcs. The Voronoi data model demonstrates the ease of performing this operation. The algorithm takes each object from a selected set and collects its Voronoi edges into a list. For each Voronoi edge (bounded by two Voronoi vertices counterclockwise), its intersections, if there are any, with the line (or circular arc) coinciding with the perspective buffer boundary, are calculated. Two consecutive intersections constitute a buffer segment. If a Voronoi region concerns a line segment, the buffer boundary is a straight line segment, otherwise, it is a circular arc. It is possible that a buffer boundary is composed of non-consecutive segments. This can be observed in that for the first intersection calculated, if the second one is not obtained with the last Voronoi edge on the list, the buffer boundary will be broken after the second intersection.

Figure 3.35 is an example of the buffer zoning process. To calculate the buffer boundary for line segment *s*, starting with the perpendicular Voronoi edge at the right end, the first intersection is obtained. The second intersection is found with a parabolic Voronoi edge. A segment of the buffer boundary is drawn as a straight line between 1, and 2. Since intersection 2 is not on the last Voronoi edge for *s*, the buffer boundary is broken between intersections 2 and 3. The last segment is found in between intersections 3 to 4.

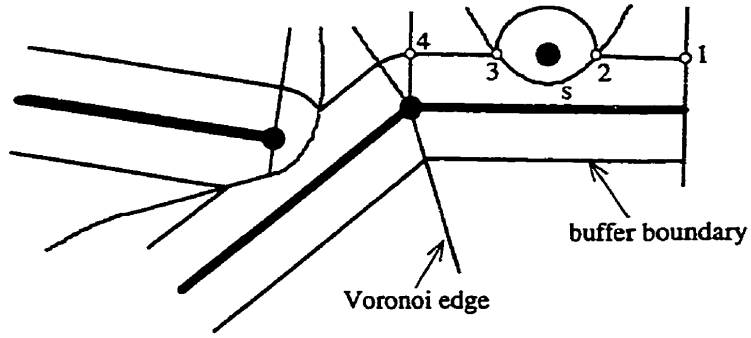


Figure 3.35 Buffering by calculating intersections with Voronoi edges

Polygon Overlay

A polygon is a spatial object carrying certain attributes. In current GIS practice, all polygons are modelled in one thematic layer if they carry the same attribute type. Frequently required in a synthesized spatial analysis is information concerning areas carrying combined attribute values of two or more attribute types. The process used to find those areas is called *polygon overlay*. Overlaying multiple layers of polygons to calculate areas with some combined attribute values presents a long-standing challenge to GIS development. All intersections of line segments forming polygons in one layer must be calculated with respect to line segments of polygons in other layers in current vector GIS. This process is not only algorithmically difficult, but also has proved to be error-prone. The kinematic incremental Voronoi data model approaches this problem in a very different way. The key idea is that the overlay process can also be done incrementally. To illustrate the Voronoi approach, we use thematic maps consisting of collections of polygons.

Let $M_1(A) = \{P_{11}(a_1), \ldots, P_{1n}(a_n)\}$ be a map object consisting of a set of polygons $\{P_{1i}(a_i)\}$ partitioning M_1, $1 < i < \infty$, $a_i \in A$, and A is a set of attribute values mapped to a well defined attribute domain. Define $L_1 = \{l_{11}, \ldots, l_{1k}\} \subset M_1$, $2 < k < \infty$, to be a set of line segments forming boundaries of the polygon set. Since each line segment in L_1, e.g. $l_{11} \in L_1$, belongs to exactly two polygons in M_1, say $P_{11}(a_1)$ and $P_{12}(a_2)$, it implies that the line segment can be treated as two directed and oriented (sided) lines. Each side of the line segment is associated with an attribute value of the facing polygon, denoted $l_{11}(a_1, a_2)$. Similarly, we can define another map object $M_2(B)$, with $\{P_{2i}(b_i)\}$, and $l_{21}(b_1, b_2)$ an example of the set of line segments in $L_2 = \{l_{21}, \ldots, l_{2k}\} \subset M_2$.

The objective of the kinematic incremental polygon overlay process, with two thematic layers, $M_1(A)$ and $M_2(B)$, is to perform a binary operation, denoted R, so that a third layer, $M_3(A\ R\ B) = \{P_{31}(a_1\ r\ b_1), \ldots, P_{3n}(a_n\ r\ b_n)\}$, $0 \leq n < \infty$, with $l_{31}(a_1\ r\ b_1, a_2\ r\ b_2)$ an example of the set of line segments in $L_3 = \{l_{31}, \ldots, l_{3k}\} \subset M_3$, is produced, where r is an instance of the binary operation R. It is assumed that both M_1, and M_2 are constructed with Voronoi diagrams. The general method is composed of the following steps (Figure 3.36):

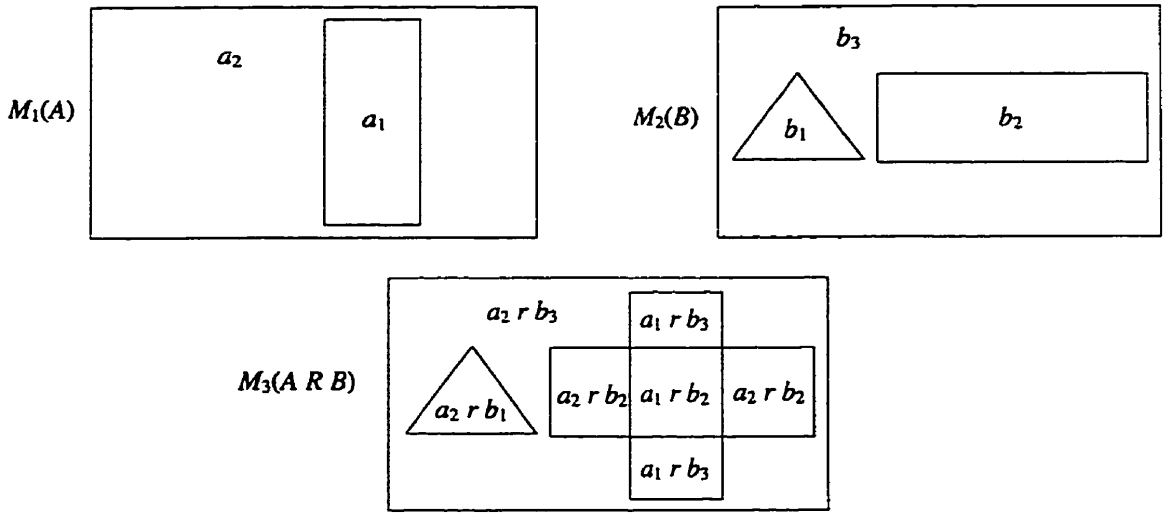


Figure 3.36 Kinematic incremental polygon overlay

1. Make a copy of M_2 and name it M_3.
2. For each line segment l_{1i} in M_1, insert l_{1i} into M_3, using the kinematic incremental method. The line segment l_{1i} may intersect with existing line segments in M_3. Preserve the associated attribute of l_{1i} in every broken segment of l_{1i}.
3. Identify polygons in M_3, using the polygon shading algorithm, and adjust associate attributes for each sided line segment in M_3.

Advantages of the Voronoi polygon overlay approach include: 1) no rigorous line intersection calculation is needed, except as part of the process of constructing the Voronoi diagram for the combined map; 2) the overlay process is easier to control, as events due to collisions and intersections are known to the process; 3) consequently, many problems such as those with coincident line segments, and sliver polygons can be resolved during the incremental construction; and 4) the algorithm also handles islands in M_1 or M_2 which remain islands in M_3.

3.8.4 Digital Terrain Modelling

Given a set of attributed objects distributed on a plane, one of the problems in *digital terrain modelling* (*DTM*) is to interpolate attribute values at any location within, say, the convex hull of the set, such that the attribute surface does not behave strangely. It has long been known that the behaviour of an interpolated surface depends largely on the choice of the portions (weights) of attribute values from selected objects around an interpolating location. The question of which objects should be selected as attribute contributors leads to discussion on identifying "the neighbours" of a given location. Gold [1992b, 1989], Gold and Roos [1994] argued that the objects in the adjacent Voronoi regions of an object in a Voronoi diagram, constitute "reasonable" neighbours of the object.

Based on the notion of Voronoi neighbours, an interpolation method called "area stealing" was described which obtains a weighted average attribute value for one interpolation. The basic idea (Figure 3.37), is to interpolate an attribute value in location x, where the point x

is imagined to be inserted into the Voronoi diagram. The Voronoi region for x would be "stolen" from neighbouring Voronoi regions. The weights of contributing attributes can be calculated, by computing individual rate of areas "stolen" from the neighbouring objects. This method has been extended to interpolating surfaces based on a set of points and line segments.

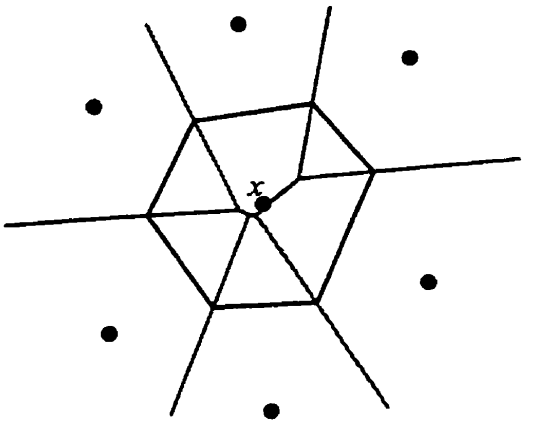


Figure 3.37 Inserting a point x steals areas from its neighbours

3.9 Summary of the Chapter

In this chapter, we presented the dynamic Voronoi data model from the following important aspects, as they are essential to the construction of the data model:

It was observed that the integrated view of spatial modelling can be found from the Voronoi diagrams. The influence vicinity analogy led naturally to the intuitive comprehension of the spatial data model. A formal definition of the ordinary Voronoi diagram was elaborated, which also anchored the geometric components of the diagram for the discourse of its properties. The properties akin to the introducing of the line segments in the object set were discussed in length. It is proved in this chapter that there exist only six types of Voronoi vertices involving an object set of points and line segments. The dual structure of the Voronoi diagram, the Delaunay triangulation was also defined, after having treated the degeneracy and its graphical representation.

Two data structures representing the Voronoi diagram, via representing the Delaunay triangulation, were discussed. More emphasis was on the triangular element data structure for it is used as the implementation structure in the research of the thesis. This chapter put a great deal of effort on the discussion of the kinematic incremental construction of the Voronoi diagram of points and line segments. The discussion includes inserting, deleting, and moving a point; inserting and deleting a line segment; the moving-in, moving-out topological events; and the detecting and handling collisions of objects. The log file structure used to preserve the history of the construction of Voronoi diagram was designed and implemented. With the log file structure, the Voronoi diagram can be reconstructed forward and backward, and be queried with regard to its temporality. It is expected that by integrating the lower level tool with tools managing changes at the conceptual level, a truly spatio-temporal GIS database engine can be developed.

GIS operations with the dynamic Voronoi diagram were discussed in detail. The algorithms for these operations cover the graph traversal over the Voronoi diagram, its dual triangulation, and the object structure; the range search; the spatial analysis, including polygon shading, buffer zoning, and polygon overlay; and the digital terrain modelling.

Through this chapter, the power and flexibility of the dynamic Voronoi diagrams in spatial modelling and analysis are basically revealed. The natural neighbourhood relationship built into the topological structure and the dynamic detecting of changes in the spatial relationship makes the dynamic Voronoi diagram an attractive choice for a spatial database model. In the following chapters, we are going to discuss the shortcomings of the primitive Voronoi diagrams of points and line segments and to seek solutions to the problems.

Chapter 4

Partitioning and Pasting Voronoi Diagrams

4.1 Shortcomings of the Voronoi Diagram of Points and Line Segments

The Voronoi diagram of points and line segments presents a powerful tool to manipulate spatial objects at the most primitive level. With the dynamic feature for updating a spatial structure, it makes an ideal *on-line* spatial database model for a dynamic GIS where the topological integrity of a map is always maintained. The impact of this feature on a spatial decision support system (SDSS) is obvious: decision models supported by the dynamic spatial DBMS obtain a *real-time* response when executing spatial and network analysis functions over collections of continuously updated map data. This is in contrast to current SDSS where the topology of a spatial database must be updated *off-line* when any modification on a spatial setting occurs. The concepts of *on-line, off-line*, and *real-time* are similar to those explained in Preparate and Shamos [1985]. An *on-line* algorithm is referred to as one that cannot look ahead at its input data. An *off-line* algorithm operates on all the data collectively. A *real-time* application relies on a special on-line algorithm that an update should be completed within an appropriate time delay.

However, used under realistic GIS or SDSS applications, the system presented here, based on the Voronoi data model, has a serious drawback: the supporting spatial data structures are not divisible into disk pages. It is assumed that, in order to operate on a map, all of the topological and geometric data structure must be loaded into computer memory. Considering the size of a typical forest GIS map with thousands of complex polygons composed of hundreds of thousands of line segments, the size of the supporting Voronoi data structure will be tremendously large. This means, as illustrated in Figure 4.1, objects and relations of the spatial data structure may occupy a large portion of the memory space, which may be allocated from different memory blocks. Notice also in Figure 4.1 that some

triangles and objects that are close in space may be addressed far apart in memory, possibly scattered in different data blocks. This is because the Voronoi diagram is generated incrementally, and hence the input of objects does not necessarily follow a regular spatial pattern. As a result, an instance of the data structure may come be characterized by a poor spatial index. This makes a simple partition or ordering of the space impossible. It follows that no matter how big a map is, all the topological and geometric objects must be loaded in memory in order to ensure the presence of the neighbourhood around any area of interest.

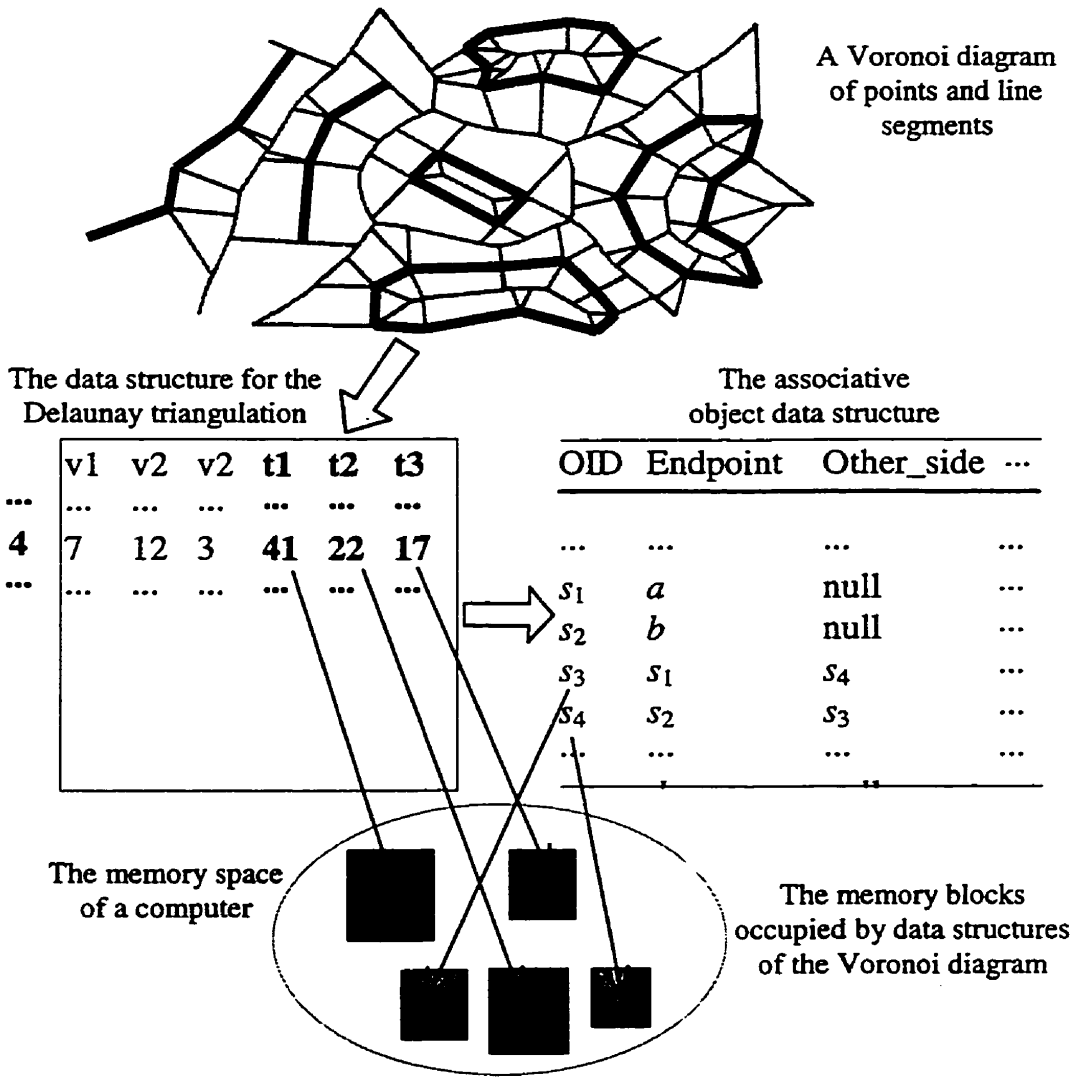


Figure 4.1 A system view of the organization of a Voronoi diagram

This drawback can raise additional problems with the GIS:

- The bulk use of computer resources lowers the overall performance of the system. It may well happen that a computer's memory is too small to completely hold a large map.
- The presence of unnecessarily detailed map objects (points and line segments) on an interactive display device hinders operators from concentrating on the more important properties of a map.
- Managing spatial objects in a non-splittable fashion fails to meet the requirements for modern data structures which need to be dynamic, hierarchical, and with varying levels detail.
- The mismatch between spatial adjacency and memory contiguity reduces the efficiency of spatial searches and presents difficulties in geometric index structuring.

The assumption of memory residence for all geometric data and data structures is common in the community of computational geometry. Although a few attempts have been made to design algorithms which process sets of geometric objects and keep at any time only a small part of their data in main memory (e.g. [Szymansky and van Wyk 1983; Ottmann and Wood 1986; Güting and Schilling 1987]), they are not generally targeted at Voronoi diagrams. The main stream of research on Voronoi diagrams and Delaunay triangulations today, including the incremental approach, still maintains the assumption of flat memory. As a matter of fact, the majority of currently practiced topological data models, such as the ones discussed in Chapter 3, have not even considered this problem, which is one of the main reasons why standard DBMS functions (as mentioned in Chapter 2) are difficult to realize in a spatial DBMS. A transition, therefore, must be made from efficient internal data structures for geometric searching problems to external geometric file structures based upon the same principles [Güting 1988].

4.2 The Objectives of the Spatial Object Condensation Technique

The rest of this chapter addresses the problems presented in the first section. The idea is not to alter the incremental construction of and other spatial operations on the Voronoi diagram, as discussed earlier, but to devise a method which partitions a larger Voronoi diagram into smaller diagrams enclosed in some designated polygon boundaries, and which pastes together smaller Voronoi diagrams (spatially) to form a larger one. This method, called the *spatial object condensation technique* [Yang and Gold 1995], needs to provide the following features to the partitioned Voronoi diagrams:

- When partitioning a Voronoi diagram, the corresponding parts of geometric objects and spatial data structures should be identified and separated from the original structures. We call each of such geometrically identifiable and topologically structured diagrams a *Voronoi map object (VMO).*
- The VMO from which smaller VMOs are partitioned is a parent, and the smaller objects are children. The parent VMO embeds the geometric boundaries of its children in its *spatial structures* (the Voronoi diagram and the Delaunay triangulation). The internal spatial structures of child VMOs are opaque to their parent and are removed from the spatial data structures of the parent. This simplifies the spatial structure while preserving an outline view of a spatial setting, and reduces the memory used for unnecessary detail.
- The boundary of a child VMO specifies the extent of the subspace and of its internal geometric objects and its spatial structure.
- Each individual VMO can be saved separately onto secondary storage, loaded into main memory, and worked with independently. By "worked with" we mean applying either construction operations, spatial searches, or analysis functions to the object.
- A child VMO can be pasted seamlessly back to its parent space with or without merging the two spatial data structures. It can also be inserted into other Voronoi diagrams, provided there is no spatial conflict. The insertion will not be exhaustive in that only the boundary of the VMO needs to be inserted into the target Voronoi diagram.

This chapter deals with the geometric and topological aspects of the technique and implementation issues involved in partitioning a set of *system objects* allocated to store and operate on elements of the data structure of a Voronoi diagram. Managing dynamic relationships between VMOs and operations upon them as individually identifiable objects is more an issue of spatial database models and will be tackled in a later chapter.

The objective concerning the geometric and topological aspects can be stated in a more formal fashion. Take a set of spatial objects $O \subset R^2$, the Voronoi diagram $V(O) \subset R^2$, and the dual topological structure, the Delaunay triangulation, $D(O) \subset R^2$, as defined previously in this thesis. The objective of the spatial condensation technique is to partition $V(O)$, within its embedding plane R^2, into a finite number of n - 1 bounded subdiagrams, $V(O_2)$, ..., $V(O_n)$, with corresponding bounded, embedding subspaces, S_2, ..., S_n, and one unbounded *left-over subdiagram*, $V(O_1)$, with its corresponding unbounded *left-over subspace*, S_1. Denote S_i°, ∂S_i, and S_i', the interior, the boundary, and the exterior of S_i, for $2 \leq i \leq n$, respectively. By the same notion, denote also S_1°, ∂S_1, and S_1', the interior, the boundary, and the exterior of S_1, respectively. The partition must satisfy the following conditions:

a) $R^2 = \bigcup_{i=1}^{n} S_i$ (the S_i are a covering of R^2),

b) $S_i^\circ \cap S_j^\circ = \varnothing$ for $1 \leq i \leq n$ and $i \neq j$ (all S_i° are mutually disjoint), and

c) $\partial S_1 = \bigcup_{i=2}^{n} S_i \cap S_1 = \bigcup_{i=2}^{n} \partial S_i$ (mutual boundaries between S_i and S_1).

Especially, denote by X_i, the set of spatial objects to be inserted into (or deleted from) S_i, $1 \leq i \leq n$, that is, a partition of a finite set of spatial objects $X \subset R^2$. We would like the partition to satisfy the following dynamic conditions:

d) $(O_i \cup X_i) \subset S_i$ for $1 \leq i \leq n$ (*dynamic subspace restriction*),

e) $O \cup X = \bigcup_{i=1}^{n} (O_i \cup X_i)$ (the O_i are a dynamic covering of O),

f) $V(O_i \cup X_i) \subset S_i$ for $1 \leq i \leq n$ (*dynamic geometric integrity*),

g) $V(O \cup X) = \bigcup_{i=1}^{n} V(O_i \cup X_i)$ (the $V(O_i)$ are a dynamic covering of $V(O)$),

h) $D(O_i \cup X_i) \subset S_i$ for $1 \leq i \leq n$ (*dynamic topological integrity*), and

i) $D(O \cup X) = \bigcup_{i=1}^{n} D(O_i \cup X_i)$ (the $D(S_i)$ are a dynamic covering of $D(S)$).

It is understood that only the unbounded subspace, S_1, is the parent of all children, S_i, for $2 \leq i \leq n$, which are bounded subspaces. The parent subspace is "left-over" with n - 1 holes after n - 1 children are partitioned. The interiors of all subspaces are disjoint, i.e. condition b), and all subspaces are connected via the mutual boundaries with the parent subspace, i.e. condition c). Conditions d), f), and h) stipulate that partitioned subsets of objects, the Voronoi diagrams, and the Delaunay triangulation are all dynamically restricted within the corresponding subspaces. By the concept of "dynamic geometric and topological integrity" we mean that the Voronoi diagrams and Delaunay triangulations need to be completely definable (and modifiable) within each subspace with respect to dynamic sets of objects embedded therein. The unions of all subspaces, dynamic subsets of objects, Voronoi diagrams, and Delaunay triangulations need to come to their counterparts before and after partitioning, as indicated by conditions a), e), g), and i).

It should be noticed that in the above formal conditions, we have deliberately neglected a few "boundary conditions" between subsets of objects, Voronoi subdiagrams, and Delaunay subtriangulations. This is because we have not specified what might constitute the boundaries of partitions. We now turn to discuss the boundary matter first, and come back to formal boundary conditions later. For the sake of simplicity, we denote the partitioning of a $V(O)$ into a finite k of subdiagrams as a *k-partition*.

4.3 Partition Boundaries

When discussing what might constitute appropriate boundaries where partitions are made, we first look intuitively at a Voronoi diagram $V(O)$ on a piece of paper (Figure 4.2), and are prepared with a pair of scissors. It seems natural that a partition can be made by cutting the diagram along some existing geometric framework, say, the Voronoi edges. By checking the result of this method with the conditions speculated in the previous section, it is obvious that all but the last conditions stated are readily satisfied by cutting $V(O)$ into $V(O_1)$ and $V(O_2)$ along the designated Voronoi edges. However, the Delaunay triangles crossing the boundary will be cut, which violates the integrity of the topological structures for both partitioned subsets.

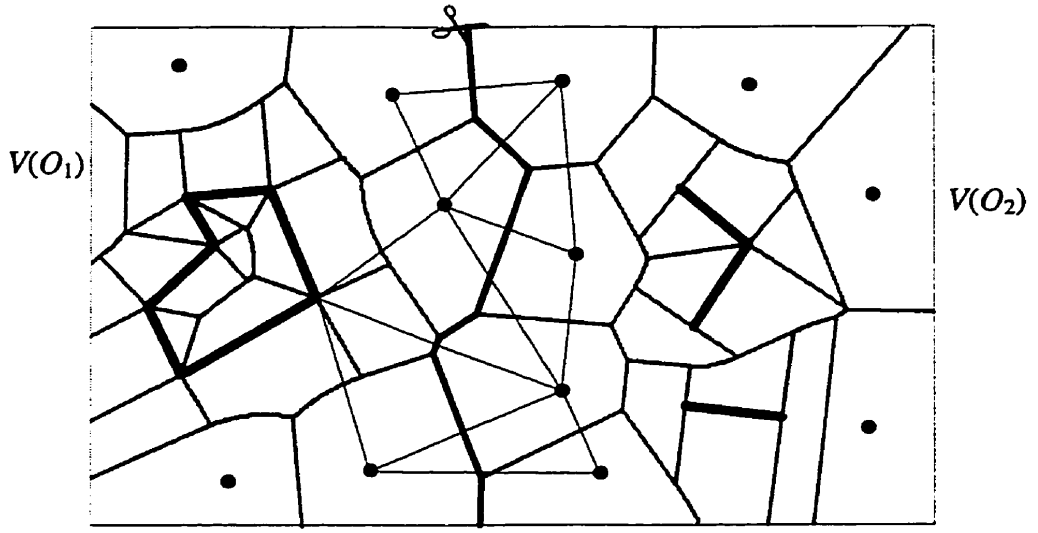


Figure 4.2 Cutting a Voronoi diagram $V(S)$ on paper

The problem of violating the last condition would not be a serious one if the Delaunay triangulation did not play a critical role in managing objects and Voronoi diagrams embedded in space, as in the example of drawing a Voronoi diagram with a ruler and a compass on paper. Also, it will not work with the computer representation relying on the Delaunay triangulation. To attempt to solve this problem, we used our imagination.

Imagine that with a retriangulating algorithm, ignoring the complication involving the boundary conditions, the cut triangles at both subspaces could be modified to be complete. One way of doing this could simply discard those cut triangles from both divided diagrams. This solution can provisionally bring other problems. The first concerns the dynamics of both subspaces. If a point $x \in S_1$ is inserted close to the boundary of S_1 and S_2, the triangles near the boundary in S_1 have to be adjusted. This consequently alters the partitioned Voronoi edges on the boundary of S_1, which violates conditions g) and i).

Naturally, one can think of using connected Delaunay triangle edges as the partitioning boundary. While cutting along triangle edges preserves the integrity of the computer representation of a Voronoi diagram (admitting duplicated representations of vertex objects on the boundary), condition f) is violated in that the Voronoi edges crossing the boundary will be cut. Again both conditions g) and i) are difficult to maintain when considering the dynamics of the object sets. However, if one emphasizes the integrity of the computer representation and admits overlapping of the Voronoi diagrams near the boundary, condition i) can be ensured by specifying that the boundary triangle edges never be switched. We will come back to this possibility after we have examined another more natural kind of boundary.

The kind of partition boundary that comes to mind more naturally consists of the components of polygonal objects. We treat a partitioned polygon as a *container object* within which other types of spatial objects may exist. Polygons as containers are frequently seen from the hierarchical structure of spatial arrangements. For example, the polygon representing the territory of a country can contain territories of provinces which further contain regions of counties, areas of municipalities and political districts, and so on. In other geographical applications, polygons can be containers representing the aggregated properties of some spatial phenomenon. Examples of aggregated polygons include forest land diversified with polygons representing different tree species or groups of trees at different ages; crop land with variations of vegetation, etc. No matter what may be contained inside, a noticeable characteristic of all such polygons is that their shapes are

naturally irregular. This contrasts with the current computerized practice of traditional cartographic mapping where the properties of maps are all contained in rectangular frames which spatially delineate boundaries of geographic databases. The choice of partitioning a large map based on the natural boundaries of container objects is advantageous in this respect.

The initial attempt of partitioning a Voronoi diagram along container polygon boundaries can be illustrated from the following two diagrams. In Figure 4.3 the Voronoi diagram and the dual Delaunay triangulation for a small collection of polygons are present. Along polygon boundaries, Delaunay edges and Voronoi edges are aligned. As is shown in the diagram, although the boundaries of those polygon are closed loops of line segments, the underlying structures are not necessarily simple. To simplify the description, we assume that they are simple closed loops, i.e. every node in a loop is connected by two line segments only. The partitioning method described later does not rely on this assumption. Now we choose to partition the container polygon with the closed boundary. The separation of two subspaces, S_1, and S_2, by partitioning the enclosed polygon region along the simple boundary is immediately guaranteed by the Jordan curve theorem, which says that a simple polygon boundary divides the Euclidean plane into two connected regions, the interior one is bounded and the exterior one is unbounded. Both regions have the same boundary as frontiers. To see that the Voronoi diagram and the Delaunay triangulation are also readily separated by the simple polygon boundary, we need to take a micro-view of both structures along the boundary.

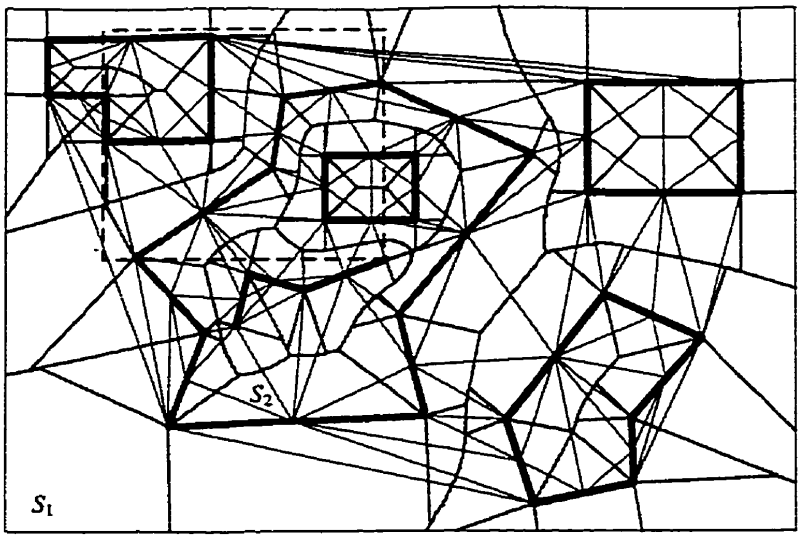


Figure 4.3 The Voronoi diagram and Delaunay triangulation of a map

An enlarged view of one part of the diagram from Figure 4.3 is shown at the left of Figure 4.4. Both Voronoi and Delaunay edges are connected on the container boundary. Some Delaunay edges are entirely on the boundary. The semantics (in the Voronoi sense) of either kind of edge can be easily interpreted based on their respective definitions.

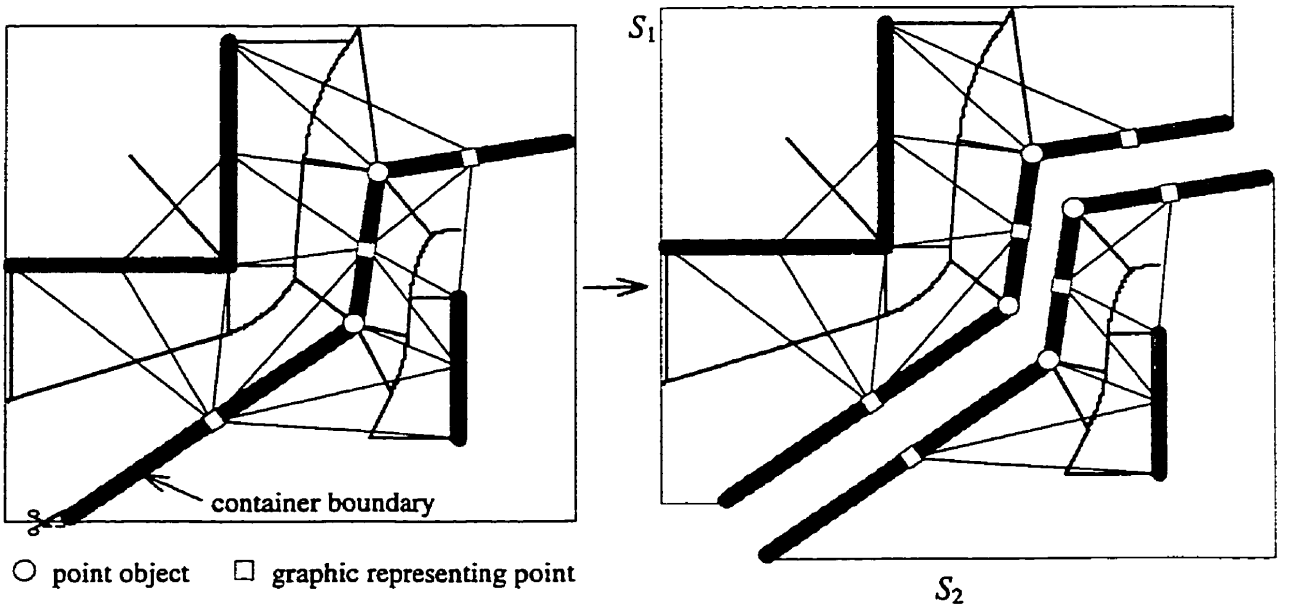


Figure 4.4 A micro-view of the Voronoi and Delaunay structures near partition boundaries

For example, any Delaunay edge between an endpoint and a graphic representation point depends on the existence of the Voronoi edge (possibly degenerated to zero length) between the endpoint and the interior of the line segment. Likewise, any Delaunay edge between two graphical representation points on the boundary corresponds to the bisecting Voronoi edge of the two line segments. Nevertheless, by the definitions of both the Voronoi diagram and the Delaunay triangulation, neither Voronoi nor Delaunay edges extend from the interior of the container to the exterior, or vice versa. Therefore, by preserving the mutual boundary of both partitioned subsets, the integrity conditions f) and h) are automatically ensured. This is shown at the right of Figure 4.4, where no Voronoi or Delaunay edges are cut by the partition.

The next concern is the dynamic behaviour of the partitioned subspaces on both sides. The question is whether changes made to the spatial structures (the Voronoi diagram and the Delaunay triangulation) in one subspace would affect the spatial structures in the subspace on the other side of the partitioning boundary. It is understood that changes to the spatial structures are caused by construction operations such as inserting, deleting, and moving objects in the object set. If these construction operations are applied only to objects in the interior of a bounded (or an unbounded) subspace, it is immediately clear that any changes in the spatial structures of the subspace will not affect those of other subspaces. The reason is that the boundary of the subspace is the outmost neighbour of any object in the subspace, which precludes the neighbourhood relationship between any two objects residing in two interiors separated by a boundary.

We summarize the above discussion with the following *boundary choice theorem*:

Theorem 4.1 (boundary choice theorem). Partitioning a Voronoi diagram $V(O)$, together with its embedding space S, along a closed polygon boundary, $B \subset O$, into $V(O_1)$ and $V(O_2)$, together with corresponding subspaces S_1 and S_2, satisfies all conditions specified in Section 4.2, provided that the dynamic subsets of objects, X_1 and X_2, belong only to the

interior of their corresponding subspaces, that is $X_1 \subset S_1^\circ$, and $X_2 \subset S_2^\circ$. The partition boundary is mutual to both S_1 and S_2, that is $B = \delta S_1 = \delta S_2$.

It is obvious that this theorem can be extended naturally to a k-partition.

4.4 The Implementation of the Partition with the Data Structure

After choosing to use the container polygon boundary to partition the Voronoi diagram and its embedding space, we now proceed to see how this partition can be done with the data structure of the Voronoi diagram stored in a computer. The objectives here are: to identify all geometric and topological components of the data structure for each subspace; to separate the child subdiagram from the original instance of the data structure; and ultimately, to save it as another instance of the data structure. The constraint for all separated instances of the data structure is that each individual instance should support the partition objectives in Section 4.2 such that the corresponding subdiagram can be operated upon independently. We remind ourselves here that the data structure of the Voronoi diagram used in this thesis is composed of two parts: the representation of the Delaunay triangulation and the representation of the associative object structure.

Denote $A(O)$ the associative object structure, and $M(A(O))$ the computer representation of $A(O)$. Likewise, denote $M(D(O))$ the computer representation of $D(O)$. Both $M(D(O))$ and $M(A(O))$ are collections of respective system objects representing triangles, points and line segments. For simplicity, we also denote $M(V(O)) = M(D(O)) \cup M(A(O))$ the computer representation of $V(O)$. An objective of the implementation of a k-partition is to partition the collection $M(V(O))$ into corresponding subsets, $M(V(O_1))$, ..., $M(V(O_k))$, such that for a proper set of functions F applied on $V(O)$, denoted $F(V(O))$, the following conditions hold:

j) $F(V(O)) \Leftrightarrow F(V(O_1) \cup ... \cup F(V(O_k))$ (function subdivision), and

k) $F(V(O_i)) \rightarrow M(V(O_i))$ is bijective for $1 \leq i \leq k$ (logical-physical transformation).

The equivalence condition j) states that the proper set of functions applied on the original Voronoi diagram can be equivalently achieved by first applying them to the collection of subdivisions and then taking the union of individual results. The mapping condition k) stipulates that the implementation of the functions on individual Voronoi subdiagrams needs to be transformable between the logical and the physical levels, and the transformation is both injective and surjective. That is, the algorithm of a spatial operation on $V(O_2)$ (logical representation of spatial objects) can be exclusively implemented on $M(V(O_2))$ (physical representation of spatial objects); inversely, any system operation on $M(V(O_2))$ corresponds to some spatial operation exclusively on $V(O_2)$.

Satisfying the above two conditions can be useful for designing parallel algorithms for spatial operations and for federated spatial database management in general. Familiar examples of the proper set of functions include construction operations, spatial searches and analysis, as discussed earlier. The "proper" modifier on the set of functions respects the "boundary conditions", especially those regarding construction operations.

Identifying and Transferring the Subset Enclosed in a Partitioning Boundary

The first step of the partitioning algorithm identifies the subset of spatial objects and their spatial structures in one designated subspace. Once identified, it can be "written" into a newly allocated memory space as a separate instance of the data structure, and later removed from the original memory space. Due to incremental construction, the logical indices of the subset of spatially adjacent objects and structures may not be contiguous, the memory blocks occupied by the subset may consequently be dispersed. Some spatial traversal technique has to be used to identify the subset and transfer the logical addresses of the members into those of the new instance of the data structure.

In our implementation, we use the polygon shading or flood fill algorithm, as discussed in Chapter 4, to traverse the topological structure (the Delaunay triangulation) in spiral-like

order. For every triangle and vertex object identified, instead of matching their original logical indices to the counterparts in the new instance, appropriate consecutive index numbers from the new instance are assigned to them. This eliminates big gaps in the memory space of the new instance between spatially adjacent objects and structures. The process of identifying and transferring logical indices is called *flood fill memory compaction*. Figure 4.5 illustrates this process where the heavy and light numbers represent indices for spatial objects and Delaunay triangles, respectively.

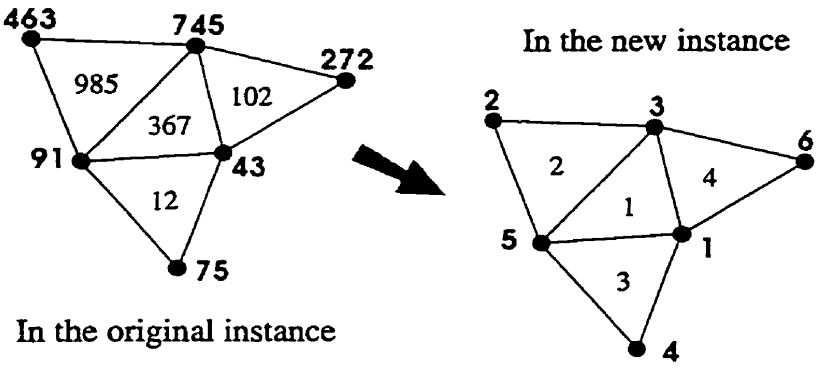


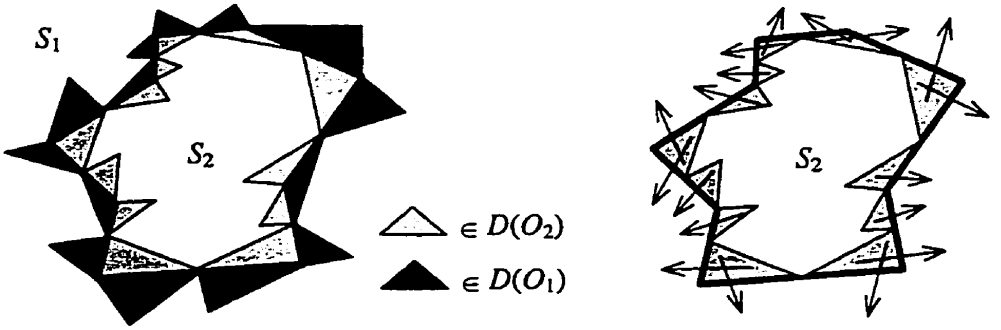
Figure 4.5 Flood fill memory compaction

Out-Pointers of the Bordering Triangles in the Data Structure

Recall that (ref. page 79) the triangular element data structure for the Delaunay triangulation not only maintains pointers to three vertex objects for each triangle, but also pointers to three adjacent triangles for each triangle in question. Maintaining pointers to adjacent triangles ensures the ability to spatially traverse the whole topological structure. In this sense, we call those pointers the *topological pointers*. For a set of triangular objects identified in one partition, say $M(D(O_2))$, it is apparent that some topological pointers in $M(D(O_2))$ will have to point to triangular objects in $M'(D(O_2))$, naming these pointers *out-pointers*. Unfortunately this violates both conditions j) and k). For example, a function on $D(O_2)$ may need to know the positions of all adjacent triangles for each triangle concerned, including the adjacent triangles referred to by the out-pointers. The function will fail when $D'(O_2)$ is removed and the subspace O_2' becomes void, because the out-pointers refer to

triangles which no longer exist in the subspace concerned. In order to solve this problem, we first identify triangles in $D(O_2)$ whose representation contains out-pointers.

For any line segment on the partitioning boundary, there can be only two triangle edges each connecting an endpoint and the graphic representation point (cf. Figure 4.4). No matter how many triangles are incident to the line segment, only two pairs of adjacent triangles share the triangle edges collinear to the line segment. Since each pair has triangles from both sides and they are face to face along the boundary line, they are called *bordering triangles* (Figure 4.6a). Referring to the data structure for the Delaunay triangulation, it is the bordering triangles that contain out-pointers to triangles out of reach (Figure 4.6b).



(a) bordering triangles (b) out-pointers in $M(D(O_2))$

Figure 4.6 Bordering triangles and out-pointers

A Flaw of Voronoi Diagrams in Dealing with Line Segments

In seeking a solution for the subset inter-referencing problem incurred by the out-pointers, an important flaw in current theory and practice of Voronoi diagrams dealing with line segments is revealed: their internal spatial structures have been either ignored or are incomplete. This finding can be made evident by the following example (Figure 4.7).

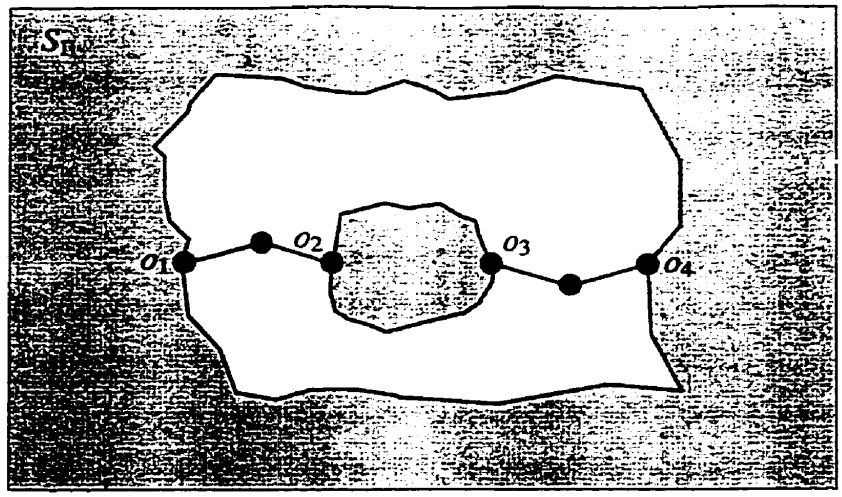
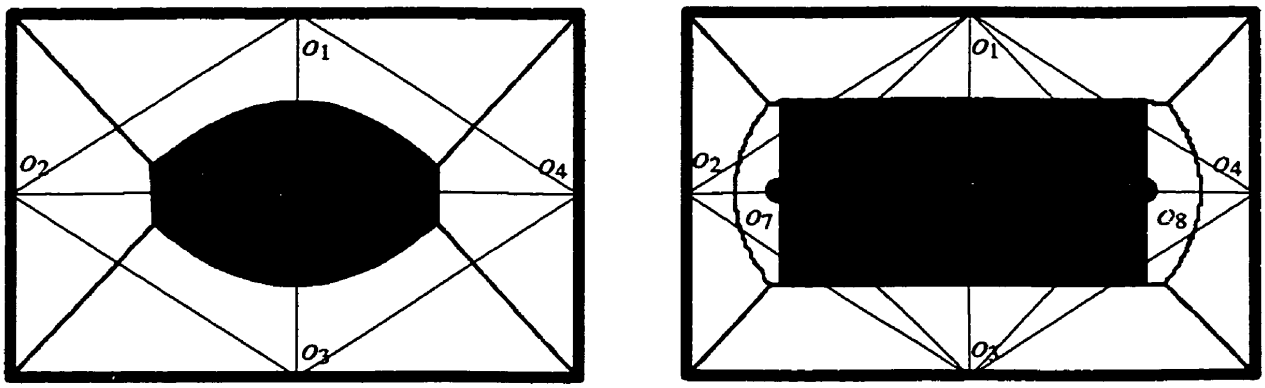


Figure 4.7 Weakly-connected subspaces

Based on pointset topology, two components in S_1 of Figure 4.7 are weakly connected via two single lines between points o_1 and o_2, o_3 and o_4, respectively. These two topological subspaces need to be traversed with a supporting topological structure. However, what would be the construction of the Voronoi diagram over a single line subspace is neither treated theoretically nor in the representation with data structures. Typically, two Voronoi diagrams would be constructed, one for each subspace. This leaves the subspace occupied by the single pass unattended. Besides, this treatment alters the topological nature of S_1 which would become unconnected. A reason for this ignorance is that most of the current studies of Voronoi diagrams consider that embedding spaces are all strongly connected and these "boundary conditions" are treated as "special cases" which are usually avoided by the main stream of research on Voronoi diagrams.

Some distinct aspects of Voronoi diagrams involving line segments have been addressed in the literature. For example, it has been noted that while the subdivision of the Voronoi region of a line segment can be conceptually analogous to that of a point, the deriving of the dual Delaunay structure around a line segment, however, is not a simple extension of that for a point (compare Figure 4.8 (a) and (b)). The orientation of a line segment must be distinguished to conform to the unique empty circumcircle condition for the Delaunay triangulation. Nevertheless, the treatment is incomplete both theoretically and practically.

Theoretically, the topological structure of the Voronoi diagram is not homogeneous on the whole plane. As demonstrated in Figure 4.8b, there exist double triangle edges between adjacent triangles, $\Delta o_1 o_5 o_8$ and $\Delta o_6 o_3 o_8$, on one hand, and a gap between Voronoi regions incurred by two objects, o_5 and o_6, on the other hand. In practice, the absence of representation of the internal topological structure of a line segment presents difficulty in dealing with, for example, the inter-referencing problem and the traversal between weakly-connected subspaces via the computer representation of their topological structures.



(a) Voronoi region and adjacent triangles around a point

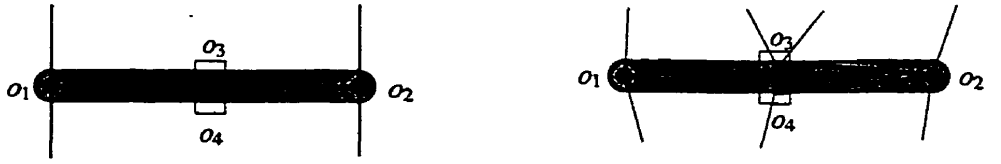
(b) Voronoi region and adjacent triangles around a line segment

Figure 4.8 Illustration of the incompleteness in handling line segments

The Completion of the Spatial Structures Within a Line Segment

The idea of completing the spatial structures within a line segment becomes simple once the problem has been analyzed. Take a close look at the geometry of a line segment whose composition includes two topologically distinguished "half-lines". Each half-line starts from one endpoint and its orientation is given by the "right-hand" rule with respect to the other endpoint associated with the other half-line. In the associative object data structure, these two half-lines represent two distinct objects and are mutually referenced. Based on the definition of the Voronoi diagram, it is natural to amend a Voronoi edge bisecting these two half-line objects (Figure 4.9a). We call this Voronoi edge the *Voronoi in-line edge* to distinguish it from other Voronoi edges. Accordingly, amend the two triangles inside the

full line segment to correspond to the Voronoi edges between the four connected objects which together describe a full line segment (Figure 4.9b). We call these the *critical triangles* because they serve as the transitional elements between interrelated subspaces.



(a) the completion of the Voronoi in-line edge for a full line segment

(b) the completion of the critical triangles for a full line segment

Figure 4.9 The completion of the Voronoi in-line edge (a), and the critical triangles (b)

The completion of the spatial structures within a line segment complies with the definitions of both structures. It is understandable that the circumcircle of a critical triangle has a zero radius and its centre coincides with the endpoint vertex. These amendments are rather more conceptual than geometric. They are nevertheless vital to solving the two problems mentioned above. The role of the critical triangles will be made more evident as the discussion continues.

Resolving Unreferenced Out-Pointers

We now come back to solve the inter-referencing problem: the *out-pointers* in one subset of system objects refer to triangular objects in other subsets. With the completion of the critical triangles, the bordering triangles no longer contain *out-pointers* because these have been transferred or contained within critical triangles on the partitioning boundary. Figure 4.10 shows the *out-pointers* in the critical triangles on the boundary inter-relating two connected subspaces.

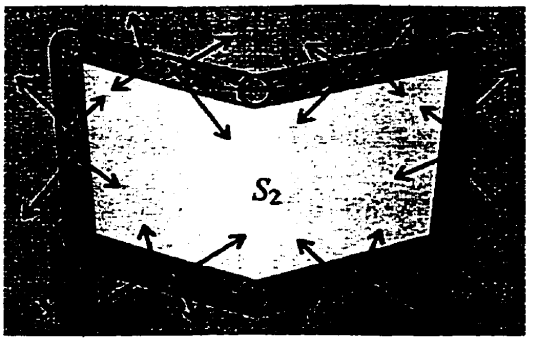
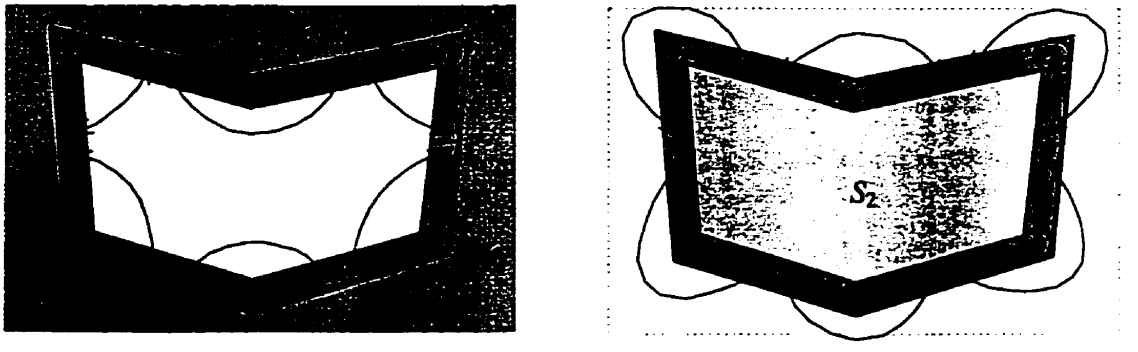


Figure 4.10 Out-pointers in critical triangles
inter-relating two subspaces

After $V(S_2)$ has been removed, the *out-pointers* from $M(V(S_1))$ cannot be resolved once they are used to access triangles in $D(O_2)$. Our solution is to replace the reference of an *out-pointer* in one critical triangle with the other critical triangle incident to the same endpoint. This is shown in Figure 4.11a as *out-pointers* are "bent" around corners. As a result, two critical triangles incident to the same endpoint are mutually referred. Likewise, the *out-pointers* on the boundary of the other partitioned subspace are similarly resolved (Figure 4.11b). The explanation for bending *out-pointers* is that whenever a traverse is about to leave a subspace, it should immediately be directed back if and when the other subspace becomes void.



(a) Resolving out-pointers for $M(V(S_1))$ (b) Resolving out-pointers for $M(V(S_2))$

Figure 4.11 The resolution of out-pointers

Applying the method of bending *out-pointers* completes the justification for the implementation of the partition technique. The equivalence condition j) and the mapping condition k) are satisfied except for the following modification.

The Modification of the Nearest-Object Search

The *nearest-object search* over the Delaunay triangulation uses the simple walking algorithm [Green and Sibson 1977; Gold 1977] to traverse the triangular network. It starts from any known triangle and walks towards a triangle connected to an object which is the closest to the target location. Figure 4.12 illustrates such an algorithm. The arrows indicates the search path that one traverses from one triangle to another. The algorithm distinguishes the probed object and the questioned object along the search path. A *questioned object* is a triangle vertex whose closeness to the target location is examined. A *probed object* is a questioned object, which is found closer than the previously probed, and all its neighbours are intended to be questioned in a circular sense. The algorithm speculates that 1) when an object is questioned with respect to a probed object, the triangular path is advanced once to the adjacent one which is connected to the object in question; 2) when a questioned object becomes the probed, a new circular questioning starts from the triangle just advanced in; and 3) a complete circle of triangles (with respect to the probed object) must be performed before claming the probed object the nearest object. In the example shown in Figure 4.12, the start triangle (gray) and the probed object *a* (gray) are on top of the partial triangulation. Object *b* becomes a new probed object when it is questioned. Then a new circular search path is attempted, starting from the adjacent triangle below the grayed initial triangle. Another new probed object, *c*, is found before a circle of triangles is formed. The search is terminated by declaring object *c* the nearest object, as a questioning circle is formed with respect the *c*.

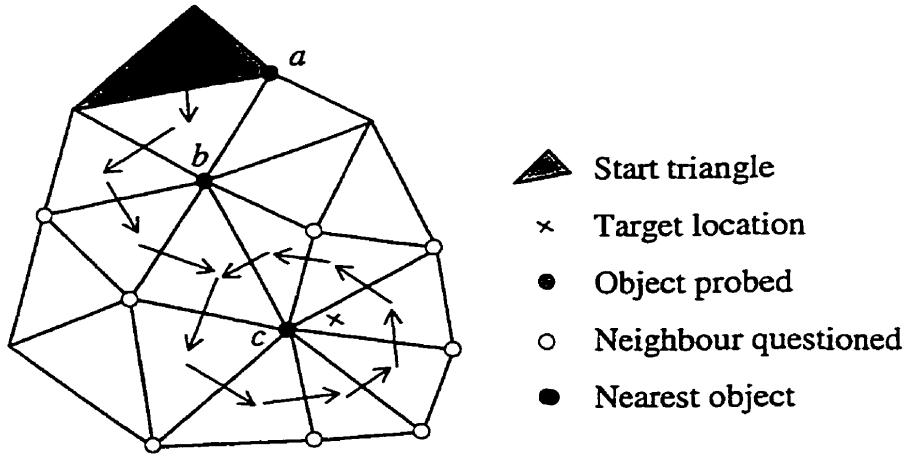


Figure 4.12 An illustration of a nearest-object search algorithm

The principles of the algorithm can be extended to the Delaunay triangulation with points and line segments. The only additional concern is about closeness test with regards to a line segment. Figure 4.13 illustrates a nearest-object search over a triangulation of points and a closed polygon in S_1. The polygon boundary is implemented with critical triangles. Notice that the search path enters the interior of the polygon.

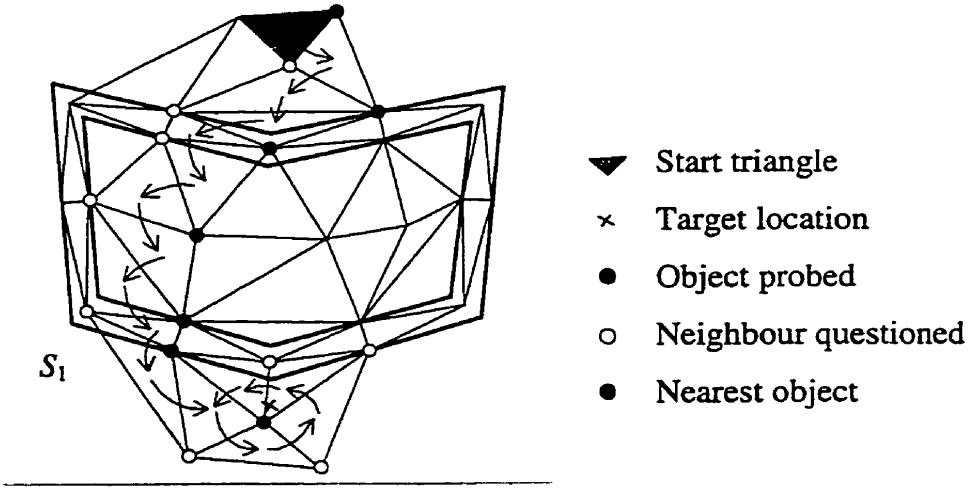


Figure 4.13 An illustration of a nearest-object search over points and line segments

The problem occurs when the interior subspace has been removed from subspace S_1. In the example shown in Figure 4.13, the search path will be broken when one traverses on the polygon boundary and is about to enter into the interior of the polygon. This effectively says that the target location is "invisible" from the start probed object in the sense that there does not exist a "straight" search path between them.

On the other hand, the application of "bending" out-pointers alters the direction that leads to adjacent triangles. Because of the bending, the nearest-search algorithm may return an incorrect answer. The reason is that the search may be terminated prematurely. Figure 4.14 illustrates this situation. Notice that the search completes a circular questioning around a probed object and reports it as the nearest-object.

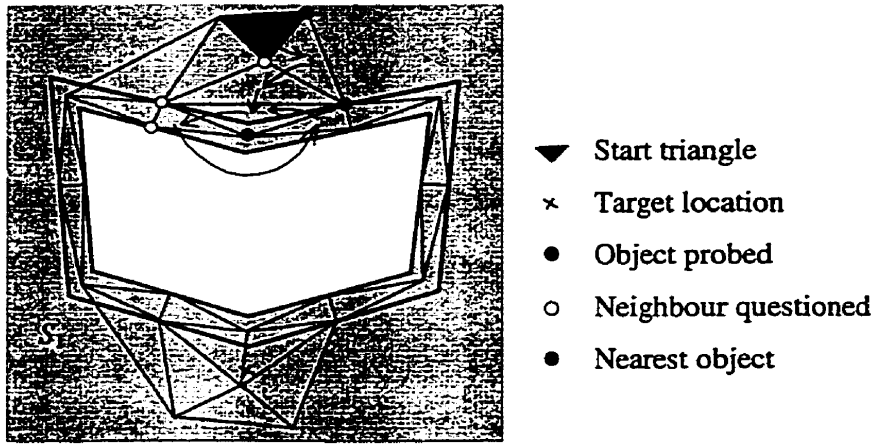


Figure 4.14 A prematurely terminated nearest-object search

The modification of the nearest-object search algorithm adds a test about the type of a triangle just traversed: Whenever a traverse walks into a critical triangle while probing an object on the partition boundary, it should never uses the bent out-pointer to walk back. Instead, the traverse should proceed to question other boundary objects in counterclockwise order, extending the triangular search path as it proceeds. The traverse continues like this until one of three cases occurs: 1) all objects on the boundary have been questioned; 2) a new probed object on the boundary is found; or 3) a probed object in the interior of the

search space is found. In the first case, a closed search path around the boundary is complete and the probed object on the boundary is the answer. For the second case, the walk continues until either case 1) or 2) or 3) is encountered. In the third case, the walking path is away from the boundary and the search is back to the normal algorithm. In the example in Figure 4.15; the traverse has encountered case 2) a few times before case 3) is found. The algorithm finally reports a correct answer.

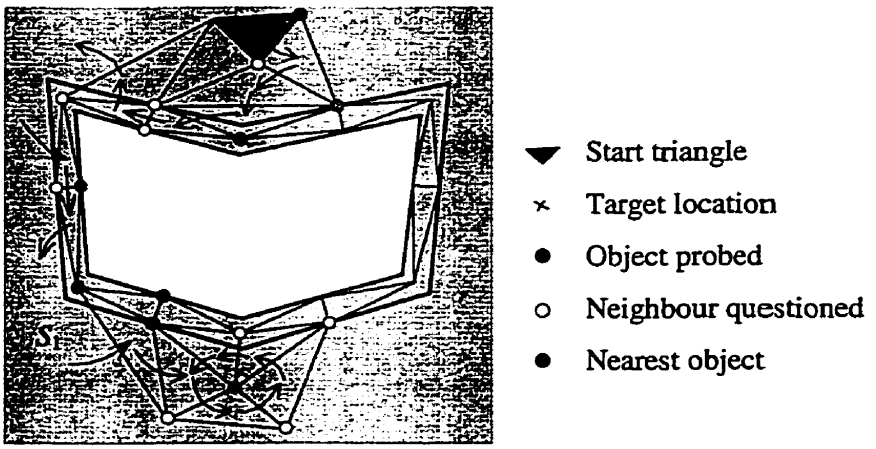


Figure 4.15 The nearest-object search after the modification

The topological concept behind the modification of the algorithm is clear: The hole occupied by the partitioned subspace is treated topologically as a special point object in the left-over search space. It is special because it has a boundary and a geometry, and its interior is condensed to a point in the left-over subspace - the very reason we call it a "condensed object". Therefore, once a boundary component of the condensed point is found closer to a target point, the search needs to complete the "circular path" around the point before it confirms the answer. We call the condensed objects "*point equivalent classes*" in the left-over subspace in a topological sense.

Traversing Weakly-Connected Components

Extending the "*point equivalence*" concept to weakly-connected components in a subspace, the problem of traversing the whole subspace is immediately solved. Based on the modified traversing algorithm for the nearest-object search, any two points in a subspace, weakly-connected or not, can be traversed by a continuous search path (the left illustration in Figure 4.16). Denote $\{o_5\}$ and $\{o_6\}$ the two equivalence points representing the holes in S_1, then the search space is topologically represented as the one at the right of Figure 4.16. It is no different than the ordinary Delaunay triangulation of an Euclidean plane.

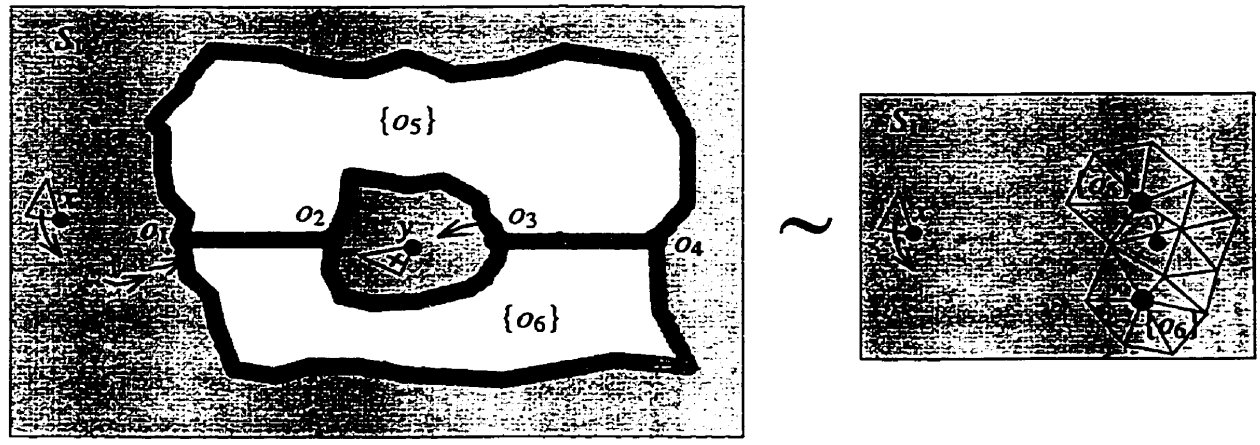


Figure 4.16 Traversing weakly-connected components and the topological equivalence

The Pseudocode of the Algorithm

The algorithm for partitioning $V(O) \subset S \subset R^2$ into $V(O_1)$ and $V(O_2)$ is summarized in the pseudocode procedure named Condense. For simplicity, we assume there exist two procedures: Identify_bound, which identifies a container polygon in S to be partitioned and returns the list of line segments in O, named *bdr*, on the partitioning boundary, in counterclockwise order; and Release M_1/M_2, which releases the part of the memory space in $M(V(O))$ originally occupied by $M(V(O_2))$. Two subprocedures named Partition S_1/S_2 and Compact D called by Condense will be expanded further.

Procedure Condense ($M(D(O), A(O))$)

Begin

1. Name $M(D(O), A(O))$ to be $M(D(O_1), A(O_1))$, short for M_1, the left-over instance of the data structure.
2. Allocate memory space for $M(D(O_2), A(O_2))$, short for M_2.
3. Identify_bound(M_1, *bdr*).
4. Partition $S_1/S_2(M_1, bdr)$. In this step, critical triangles in $D(O_1)$ are embedded for each element in *bdr*. All critical triangles are flagged with the logical indices of the corresponding critical triangles to be embedded in M_2. These indices can be calculated. At the same time, the *out-pointers* in critical triangles are "bent".
5. Compact $D(M_1, M_2, bdr)$. This step first embeds critical triangles on the boundary in M_2. This is followed by duplicating the partitioning boundary in M_2. Each critical triangle is flagged with the index of the corresponding critical triangle in M_1. Finally, starting with a given triangle in M_1, the flood fill algorithm is applied, which transforms $D(O_2)$ and $A(O_2)$ in M_1, bounded by the partitioning boundary and flagged critical triangles, to a contiguous space in M_2.
6. Release $M_1/M_2(M_1, M_2)$.

End.

Procedure Partition $S_1/S_2(M_1, bdr)$

Begin

1. Proceed with each pair of line segments in *bdr*.
2. Collect bordering triangles for the pair, then put them in a temporary variable of structure type *bt*.
3. Create four critical triangles and insert them into the pair of line segments, referring to the information in *bt*.
4. Flag each critical triangle with calculated logical triangular indices which will be used for the critical triangles in M_2.

5. Modify *out-pointers* in the critical triangles. Care must be taken with the first critical triangle, because its preceding critical triangle is the last one whose logical index will be calculated when processing the last line segment in *bdr*.
6. Take another pair from *bdr*, while repeating Steps 2 through 5 until all line segments in *bdr* are processed.

End

Procedure Compact D(M_1, M_2, *bdr*)

Begin

1. For each pair of line segments in *bdr*, create critical triangles and insert them into M_2. The vertex pointers of each critical triangle are created and inserted into M_2. The logical triangular indices created are identical to the ones calculated in Partition S_1/S_2.
2. Modify the *out-pointers* in the critical triangles. Care must be taken of the first critical triangle, because its preceding critical triangle is the last one whose logical index will be calculated when processing the last line segment in *bdr*.
3. Flag each of the critical triangles with the indices of the corresponding critical triangles in M_2.
4. Flag each of the boundary objects with the indices of the corresponding boundary object in M_2.
5. Repeat Steps 1 through 4 until all line segments in *bdr* are processed.
6. Apply flood fill. Start from any triangle *oldt* in M_1 which lies in the interior of subspace S_2. Create a new triangle named *newt* in M_2. Flag *oldt* with *newt* and push *oldt* into the priority queue named *pq*.
7. Pop one triangle *oldt* from the queue. Obtain the flag from *oldt* and name the triangle indexed by the flag *newname*.
8. For each adjacent triangle, *adjt*, of *oldt*, both in M_1, in counterclockwise order:

a) If *adjt* is not flagged, create a new triangle *newt* in M_2, flag *adjt* with *newt*, push *adjt* in *pq*; otherwise let *newt* be the triangle indexed by the flag.

b) Fill one topological pointer in *newname* with *newt* and one in *newt* with *newname*.

c) Examine the vertex *oldv* in *adjt*, opposing *oldt*. If *oldv* is not flagged, create a new object *newv*, with the same type as *oldv*, in M_2 and flag *oldv* with *newv*; otherwise let *newv* be the object indexed by the flag.

d) Fill one object pointer in *newt* with *newv*.

e) Examine the vertex *oldv* in *oldt*, opposing *adjt*. If *oldv* is not flagged, create a new object *newv*, with the same type as *oldv*, from M_2 and flag *oldv* with *newv*; otherwise let *newv* be the object indexed by the flag.

f) Fill one object pointer in *newname* with *newv*.

9. If *pq* is not empty, go to Step 6.

10. Save M_2 as a spatial page in a secondary storage medium.

End.

Analysis of the Algorithm

Obviously, additional storage for a flag is required for each triangular element and associative object. The cost for storing flags is in the order of O(n), where n is the number of objects in $V(O)$.

The time spent for the procedure Partition S_1/S_2 includes 1) walking from a known triangular index to any triangle incident to the partitioning boundary; 2) collecting bordering triangles to each line segment in M_1; 3) inserting critical triangles in M_1, and 4) modifying *out-pointers* in critical triangles. Traversing from any triangle in $D(O_1)$ to an incident triangle of a boundary line segment costs $O(\log n)$ which is worst-case optimal, as analyzed in Chapter 3. The upper boundary is in the order of $O(n)$. Processes 2) through 4) cost a constant time for each line segment and the total is proportional to m, the number of line segments on the partitioning boundary. Therefore, the upper boundary for Partition S_1/S_2 is $O(n)$.

The Compact D procedure, to compact M_2, includes creating and inserting critical triangles in line segments on the partitioning boundary in M_2, the total of which takes $O(m)$ time; and

the flood-fill process utilizing a priority queue, for which the worst case can involve all triangles in $D(O)$ and all objects in $A(O)$. The upper boundary is in the order of $O(n)$ based on the linear behaviour analysis of the Voronoi graph in Chapter 3. Storage for the priority queue has accordingly an upper boundary of $O(n)$, for adjacent triangles in the queue covering all triangles in $D(O)$.

The following theorem summarizes the implementation of the partitioning algorithm:

Theorem 4.2. The implementation of a 2-partition over the Voronoi diagram $V(O)$, for a finite collection of n points and line segments in O, satisfying conditions j) and k), can be affected in the worst-case by $O(n)$ in both time and storage.

4.5 Pasting Together Voronoi Subdiagrams

In this section, we discuss the reverse process of partitioning a Voronoi diagram into subdiagrams which paste together subdiagrams into a seamless whole. Each subdiagram is constructed separately. The resulting Voronoi diagram will contain single sets of Delaunay triangles and spatial objects which are the union of the respective subsets. The union diagram can be stored either in a single instance of the data structure, or in separated instances. In this section, we deal with the first case that the whole diagram is merged both geometrically and in storage. The second case treats the union of the Voronoi diagrams as functioning geometrically and topologically as a whole but the components of the union are stored in different instances of the data structure. The dynamic interactions of the latter case will be tackled in Chapter 5.

Two possibilities are considered when pasting a partitioned subset into a Voronoi diagram. The first one pastes the partitioned subset (being bounded) back into the original left-over subspace of the Voronoi diagram, presumably the geometry of the common partition boundary has not been modified. The other possibility pastes the partitioned subset into

another Voronoi diagram, provided that there will be no spatial conflicts on the location of the boundary. For clarity, we name the subset of the Voronoi diagram to be pasted the *source set* and the Voronoi diagram into which the source set is to be pasted the *target set*. It is assumed that both the *source* and the *target sets* have, or can be transformed into, the same Cartesian coordinate units and origin in the Euclidean plane.

The algorithm of pasting a partitioned subdiagram back to its left-over subspace is basically the same as that described in procedure compress D, with the following changes: 1) the first argument should pass in the memory instance of the *target set*, M_2, and the second one passes in that of the *source set*, M_1; 2) the first five steps will not be used because the critical triangles already exist in the *target*. Since the critical triangles and the boundary line segments in the *source set* have flags which are indices referring to critical triangles and boundary line segments in the *target set*, the completion of the flood fill (Steps 6 - 10 in procedure compact D) appends the storage indices of the data structure for the *source set* to that of the *target set*. The critical triangles in the boundary may be removed, if so desired, after two subspaces are pasted together.

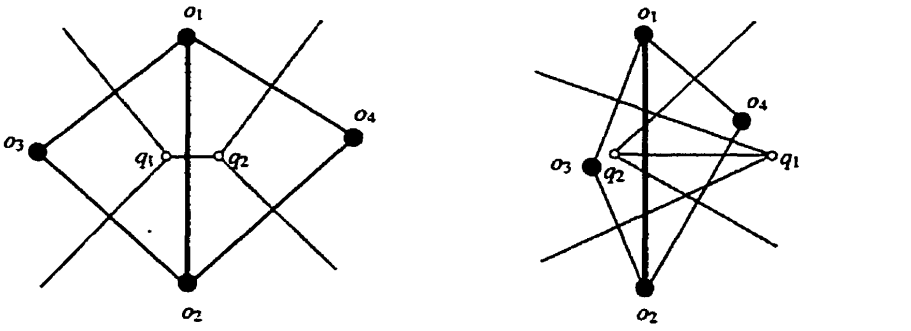
To paste a partitioned subset into another Voronoi diagram, since the partition boundary does not exist in the *target set*, it needs to be inserted into the *target set*, taking the boundary geometry from the *source set*. If no spatial conflict occurs in inserting the boundary, the geometry of the new boundary will be the same as the one in the *source set*. This process adds a closed polygon in the *target* space and the interior of the polygon is populated by Voronoi regions. The second step is to replace these Voronoi regions with the ones contained in the interior of the *source set*. Since the flags in the critical triangles of the *source set* are indices referring to critical triangles in the left-over subspace, they are unknown to the *target set*. Additional operations have to be taken to match topological pointers in two subsets. This can be done by 1) collecting bordering triangles in the exterior of the new polygon in the *target set*; and 2) passing the information about the bordering triangles to the *source set* to modify the flags in the critical triangles such that the modified flags are indices of the corresponding bordering triangles in the exterior of the new polygon.

After this, the modified procedure compact D (as the one used for the first case) can be used to paste together two instances of the storage structure.

4.6 Partitioning a Spatial Structure Along Designated Triangular Edges

In geographical applications, it may well happen that geometrical objects embedded in space are not contained in any polygons. A good example is the digital terrain model with sample points only. In this case, all points may be structured with the Delaunay or other triangulations. We discuss briefly in this section how the spatial object condensation technique can be applied to triangulated spatial structures. The objective of this application is to solve the more general problem that "Sometimes the data set in the digital terrain model is too big to be loaded into the main memory".

The idea of partitioning a triangulated data set into storage-independent but geometrically and topologically integrated subsets is the same as that of partitioning a Voronoi diagram. However the partitioning boundary with a triangulation is not a polygon object but the designated triangular edges in a closed loop. To achieve storage and functional independence, critical triangles will be implemented in the boundary edges, which will become "fixed" (they will be kept unswitched) in the dynamic operations on individual subsets after the partition. In the discussion, we deliberately avoid using the dual Delaunay triangulation of a Voronoi diagram. The purpose is not to assume the empty circumcircle condition for the Delaunay triangulation. If the partition does use a Delaunay triangulation, it should be borne in mind that Voronoi regions around the partitioning boundary may overlap due to the dynamic operations over each partitioned subset, as shown in Figures 4.17 a) and b). The nearest object search, therefore, cannot be assumed on triangles bordering the boundary. The properties and applications of the triangulations with some "fixed" triangle edges are part of the study of constrained triangulations (cf. Lee and Schachter [1980]; Chew [1987, 1989]).



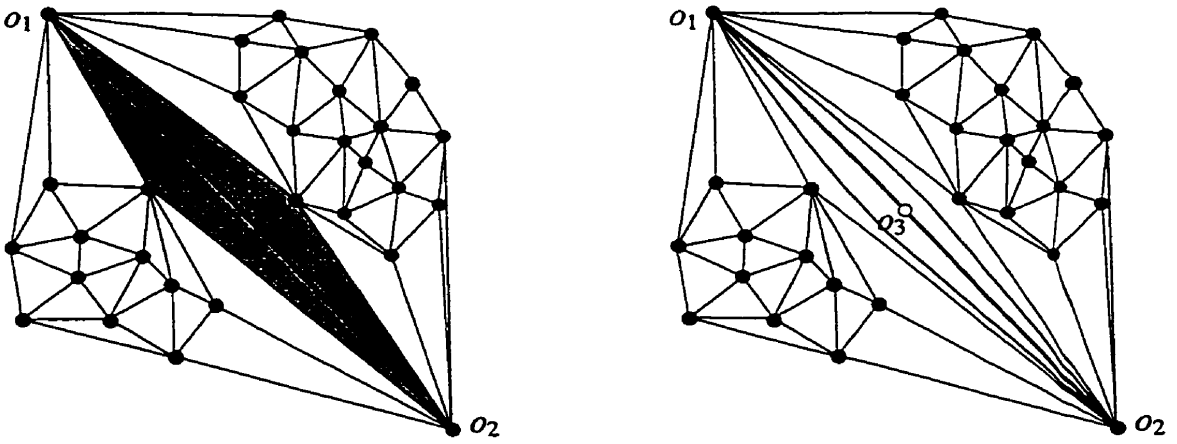
a) Triagular edge o_1o_2 is "fixed" on a partitioning boundary

b) Moving two vertices on bordering triangles causes overlapping of Voronoi regions

Figure 4.17 The loss of the empty circumcircle property on bordering triangles

With the above background, we phrase our objective for the problem mentioned at the beginning of this section: given a triangulation, with some constrained triangle edges forming a closed loop, find a method to decompose the triangulation along the closed loop, called the partitioning boundary, into two subsets such that each subset of the triangulation can be stored and dynamically operated on independently in main memory.

The method we propose generates critical triangles on the partitioning boundary. For each constrained edge between points o_1 and o_2, two bordering triangles, at both sides of the edge (Figure 4.18 a), are collected, and used to create critical triangles. It turns out that two additional triangles are sufficient to serve as the critical triangles with the same purpose as the ones embedded in a line segment. These two critical triangles can be generated by inserting one auxiliary point o_3 in the middle of the constrained triangle edge which is broken into two, o_1o_3 and o_3o_2 (Figure 4.18 b). The position of o_3 is determined by the location vector $(\boldsymbol{o}_1 + \boldsymbol{o}_2)/2$, where $\boldsymbol{o}_1$ and $\boldsymbol{o}_2$ are location vectors of o_1 and o_2, respectively.



a) Two bordering triangles at both sides of constrained edge o_1o_2

b) Two critical triangles, $\Delta o_3o_1o_2$ at the down side, and $\Delta o_3o_2o_1$ at the up side

Figure 4.18 Generating critical triangles about a constrained edge

It can be seen that there are now two constrained edges with the same two vertices. This is proper with the triangular element data structure. The two geometrically identical triangle edges belong to the components of two distinct (in topological sense) triangles, $\Delta o_3o_1o_2$ and $\Delta o_3o_2o_1$. The two constrained edges are analogous to the two oriented lines in one line segment within which the critical triangles are embedded. This piece of the partitioning boundary is now completely transformed into a "hard boundary segment" and will be saved on both partitioned subsets.

Once all boundary triangle edges are transformed, the topological connections between two subsets are implemented as *out-pointers* in critical triangles, as is shown in Figure 4.19. These *out-pointers* will be "bent" in order for the two subsets S_1 and S_2 to function independently. This can be implemented so that each of them is directed, say in counterclockwise order, to the other critical triangle incident to the same point (Figure 4.20). Therefore, the *out-pointers* in S_1 form a closed loop around the partitioning boundary clockwise (Figure 4.20a). Likewise, the *out-pointers* in S_2 form a closed loop in counterclockwise order (Figure 4.20b). The rest of the algorithm for transforming triangular and object indices is similar to that using the line segment boundary.

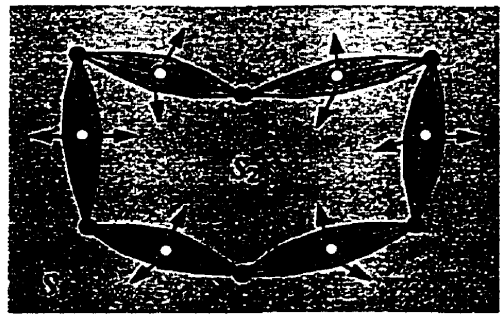
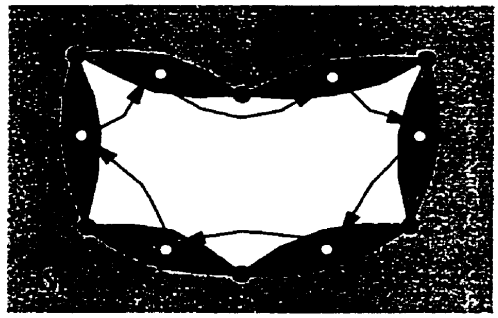
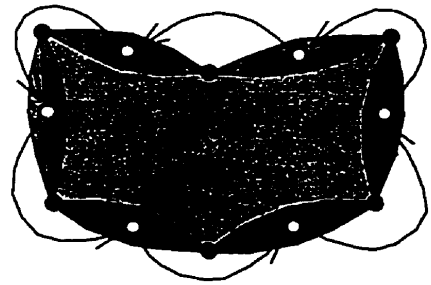


Figure 4.19 Out-pointers in transformed constrained triangle edges



a) Resolving out-pointers in S_1



b) Resolving out-pointers in S_2

Figure 4.20 Resolution of out-pointers in partitioned subspaces

Chapter 5

The Voronoi Map Object (VMO) Model

5.1 Introduction

The spatial object condensation technique decomposes the Euclidean plane, and its topological and geometric representations with data structures, into subsets wrapped independently as Voronoi Map Objects (VMO). The technique also encompasses the ability of pasting partitioned or individually created VMOs to form a larger space and representation. What has not been discussed is the managerial mechanism or structure that oversees the object condensation process, and that manages and manipulates the resulting VMOs as an integral whole. Without the complementary structure at the management level, the effort made in condensation could not be fully utilized. This chapter is devoted to this matter, and will cover the following issues: 1) What is the proper structure to collect and manage the set of VMOs? 2) What are the spatial objects and relationships that can be supported and manipulated with the structure? 3) What are the operations and constraints applicable to the spatial objects maintained in the structure?

The above issues actually touch the fundamental problems in developing a contemporary spatial data model for GIS. The problems with a spatial data model span all aspects of processing spatial data, especially with the integration of geo-object designing, storing, manipulating, analyzing, viewing, and communicating. "Contemporary" may be labeled with the increasingly "luxurious" requirements from users of GIS who are demanding to handle spatial data with more flexibility. Ideally, any interesting data in a large spatial database are modelled, managed, and viewed as an operable unit or object. The change of focus back and forth between objects should be smooth and transparent, and the database should allow both outlined and detailed views. From the systems point of view, these requirements imply that spatial data be geometrically and topologically structured in a

hierarchical fashion, with varying levels of detail. Each object in the hierarchy accommodates a region of arbitrary polygonal shape which again contains both connected and disconnected points, lineal objects, and smaller regions of different complexity; and the objects at each level correspond permanently to disk pages.

Most traditional spatial data models based on a hybrid architecture are composed of a planar graph topological data model and a geometric indexing data model. They can hardly satisfy these increasingly sophisticated needs, the reasons having been analyzed briefly in Chapter 2. In recent years, research on spatial data models for GIS has been moving towards developing more generic spatial objects with enhanced expressive power. The enhancement has been concentrated on formal descriptions of spatial objects which can be incorporated with their temporal aspect and with multiple resolutions.

For example, based on the work by Worboys [1992a], classes of spatial objects embedded in the Euclidean plane are formally defined, together with the operations on them, using the mathematical tool of simplicial complexes. The formalism is extended to model spatio-temporal (ST) objects [Worboys 1992b]. The spatial objects constitute an indispensable component within a larger frame of geo-objects which is being attempted with the object-oriented approach [Worboys 1994].

In a multiple scaled representation, Bertolotto et al. [1994] report a HPEG (Hierarchical Plane Euclidean Graph) for a multiresolution representation of a region. The HPEG recursively decomposes regions into smaller ones. Region boundaries at one level are simplified by creating ε-homotopys with line generation algorithms, where ε specifies the radius of a band convoluted from a chain of edges. Each smaller region in HPEG is a PEG (Plane Euclidean Graph) [de Floriani et al. 1993] whose features are refined with smaller horizontal error. In order to support navigation in the hierarchy of PEGs, the boundary information of a PEG must be recorded and the links to the direct refinements of a PEG must be maintained. The HPEG provides a way of browsing a map at different levels of resolution. The formalism, based on the cell complexes, for this map model is provided by Puppo and Gettori [1995]. The implementation of the HPEG is based on the DCEL

encoding structure. A hierarchical geometric indexing structure is suggested in the implementation in order to speed up data access.

We propose, in this chapter, a hierarchical data model whose node objects come from the condensation technique described in the previous chapter. This data model is aimed at managing a set of meaningful map objects which together constitute a federated spatial database, possibly distributed in a computer network. Each component of the hierarchy accommodates a subspace where a collection of spatial objects is embedded and structured both geometrically and topologically. In addition, each node corresponds to a disk page and can be loaded into memory to work independently of other node objects. Dynamic GIS operations on the spatial objects in the hierarchy are inherited from what has been discussed on the Voronoi data model in the previous chapters of this thesis. These operations can be defined within each node object to comply with object-orientation technology. A partial discussion of this data model is in Yang and Gold [1996].

The VMOs created by the condensation technique proposed in Chapter 4 naturally form a hierarchical relationship where the object from which a partition is made is the *parent* and the newly partitioned object is the *child*. A parent can have a number of children and each child itself can have children when the condensation is recursively applied to the child VMO. When a complex polygon is condensed, the partitioning boundary is duplicated in both parent and child objects. At the parent level, the subspace occupied by a child VMO is enclosed by a simple polygon whose composing detail is suppressed. Topologically, the enclosed subspace is equivalent to a hole in the left-over subspace.

We first define spatial objects accommodated by the proposed data model. This is followed by the specification of valid topological relationships between spatial objects. These topological relationships are supported by the Voronoi diagram embedded in the largest spatial object, the VMO. The representation of the data model is a tree of VMOs.

5.2 The Geometric Object Classes

The data model provides support for the following geometric data types or object classes:

- *Points*, denoted P. A point $p \in P$ is characterized by a pair of ordered real numbers $x, y \in R$, denoted as $p(x, y) \in R^2$. The pair (x, y) is called the coordinates of p.

- *Directed and oriented line segments*, denoted *LSG*. A directed and oriented line segment $lsg \in LSG$ is defined by two endpoints $p_1, p_2, \in P$, denoted $lsg(p_1, p_2)$. Each lsg is a collection of points satisfying the following convex combinatorial equation

$$lsg(p_1, p_2): \{\lambda p_1 + (1 - \lambda)p_2 \mid \lambda \in [0, 1]\}.$$

  The line segment $lsg(p_1, p_2)$ is directed from the endpoint p_1 to the endpoint p_2. The *left-side* of lsg is defined as a set of points forming a halfplane, $LH \subset R^2$, bounded by the line T (cf. Section 3.2)

$$T = \{\lambda p_1 + (1 - \lambda)p_2 \mid \lambda \in R\},$$

  such that $\forall p_3 \in LH$, the following determinant condition holds

$$D = \begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix} > 0$$

  Likewise, the *right-side* of lsg is a set of points forming a halfplane $RH \subset R^2$, bounded by T, such that $\forall p_3 \in RH$, the determinant condition $D < 0$. The dual concept is introduced to a directed and oriented line segment, such that $\forall lsg(p_1, p_2) \in LSG$, its *dual directed and oriented line segment* is defined as $lsg(p_2, p_1)$ which is also in *LSG*. For simplicity, a directed and oriented line segment (or its dual) is called simply a line

segment, when no ambiguity arises. The left-side halfplane is called the *referencing plane* of $lsg(p_1, p_2)$.

- *Directed and oriented lines*, denoted L. A directed and oriented line $l \in L$ is composed of a sequence of connected line segments in LSG, such that each endpoint in P is incident to exactly two line segments, except possibly for two endpoints, called the *extremes* of the line. The direction of a line $l \in L$ is designated by its two extreme nodes, s and e, called the *starting* and *ending nodes*, and the line is denoted as $l(s, e)$. The orientation is defined such that walking the line from s to e, the reference plane is always on the left-side of the walking direction. The dual of a line $l(s, e)$ is $l(e, s)$. When the line starts and ends at the same node, it becomes *closed*. The relationship between the class L and the primitive classes LSG and P is $(LSG \cup P) \subset L$, indicating the fact that a line is a subset of the union of line segments and points.

- *Polylines*, denoted PL. A polyline $pl \in PL$ is composed of lines which intersect only at the extremes of lines. The intersection point can be incident to more than two lines. The directions and orientations of each composing line can be worked out based on that represented for the line segments. These properties cannot be properly applied to a polyline if it is multi-branched or complexly networked. The relationship between the class PL and the class L is simply $L \subset PL$.

- *Regions*, denoted A. A region $r \in A \subset R^2$ is the area enclosed by a closed line. The closed line is the boundary of r, denoted ∂r ($\partial r \in L$), for which no extreme can be found. The interior of r, denoted r°, together with ∂r, constitute the region object, r. For convenience, the exterior of r is denoted r'. Both r° and r' have a homogeneous *internal structure*, i.e., object composition and topology. A region is a container object where points, line segments, lines, polylines, and smaller regions can be embedded. It is also possible that a region r contains holes which represent maps, to be defined. In the case that a hole is contained in r°, ∂r and r' of r will be disconnected. The boundary of the hole is the *outer boundary* of the hole which is simultaneously the *inner boundary* of r.

- *Maps*, denoted M. A map $m \in M$ is an appropriate subset of the Euclidean plane R^2. It can be unbounded or bounded. An unbounded map is called the *universal map*. A bounded map (denoted as $[m]$) has an area enclosed by a closed boundary, denoted ∂m. The boundary is distinguished by its *internal side* and *external side*. The internal side is connected to m°, the interior of $[m]$, and the external side is connected to m', the exterior of $[m]$. The boundary of an $[m]$ can be defined either by a subset of L or by a subset of P plus some structural lines in R^2 to link points in P for a closed polygon.

  Similar to a region object, a map is also a container of all spatial objects defined above and including smaller maps in M. An $[m]$ and a region r indeed behave the same at the object level. The difference emerges only when putting them in a container level, where the internal structure of m° is hidden while that of r° is always visible. The interior of a map object in a container actually constitutes a hole in the container space. When the internal structure of an $[m]$ is homogeneous to the structure of its exterior m', it can be pasted back to fill the hole.

  The reasons for having a map object class, in addition to the region object class, are: 1) the data describing the internal structure of a map may temporarily be unavailable (such as in multi-user and networked applications); 2) the internal structure may have a different format (for multimedia representation and heterogeneous databases); and 3) for hierarchical representation and management of large quantities of geometrically and topologically structured spatial objects.

A diagrammatic representation of a bounded map object is shown in Figure 5.1 which depicts the boundary and the internal objects accommodated in the interior of the map. The grayed area is the embedding space for a set of points, line segments, lines, polylines, regions, and holes (white). The holes represent smaller bounded map objects whose internal structures are not present at the container level. The detail compositions of the contained map objects are specified at a lower level. On the other hand, the internal object composition in a region object is present and visible at the container level.

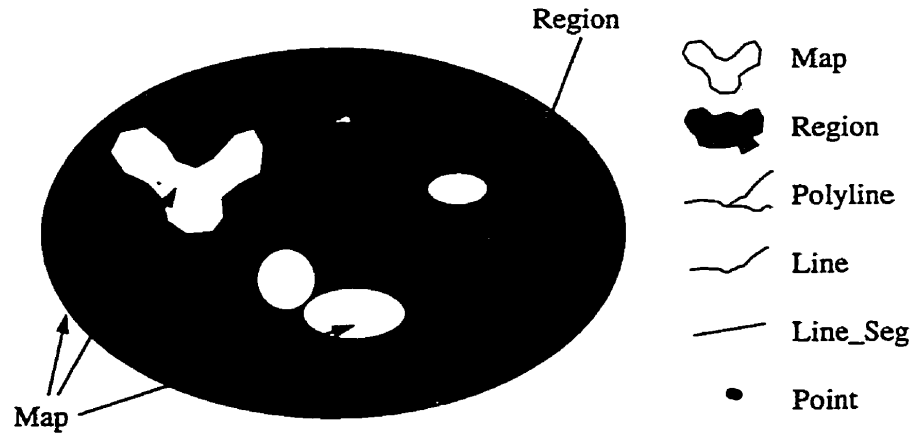


Figure 5.1 Geometric object classes

5.3 Topological Relationships of the Object Classes

Figure 5.2 illustrates an incomplete set of detail topological relationships between the geometric object classes supported by the data model. We follow the terminology of describing basic topological relationships defined by Egenhofer and Franzosa [1991]. The terminology is extended to cover some of the topological relations between regions with holes [Egenhofer et al. 1994]. Formal definitions based on pointsets for the set of spatial relationships are covered in the above mentioned two papers. The VMO data model excludes, under the static spatial database context, the general overlapping between spatial objects. Instead, dynamic operations are provided to support the overlapping process incurred in map overlay.

The shaded rectangular background represents the embedding space of a bounded map (or a region). The category of "area/area" concerns the relationships between two areal ($[m] \in M$ or $r \in A$) objects in the container. They can be disjoint, such as (a); touch at one or more common points, such as (b) and (d); or touch at one or more common boundaries, such as (c) and (e). The enclosure of the union of areal objects overlaps part of the container region. Likewise, the inner boundary of a map may be totally connected to the outer boundary of

another map (f). It should be noted that the relationship (f) is not one of overlapping: the interiors of two map objects are disjoint. The outer map object is therefore a torus. Likewise, the inner boundary of a map of the torus shape can be totally connected to the outer boundary of a region (g). In this case, the torus disconnects its exterior, i.e. the interior of the container.

In the category "area/line", an areal object and a line are concerned. A line in the interior of the container can touch a contained map at one point (h). A line enclosed in a region (not a map) touches the region boundary at one point (i). Likewise, a line can also pass through the interior of a region (not a map) and intersects the region boundary two or more (but finite) times (j). For the relation (j), if the region becomes a map object, the segment of the line enclosed in the interior of the map will not be visible at the container level (k).

The relations in the category of "line/line" are trivial. Two lines can be connected to form a longer line (l), or multiple branches of lines intersect at one common extreme point (m).

The relation (n) depicts isolated points contained in a region (the rectangle). The relations between points and lines are implicitly represented in some of the previous relations in the diagram.

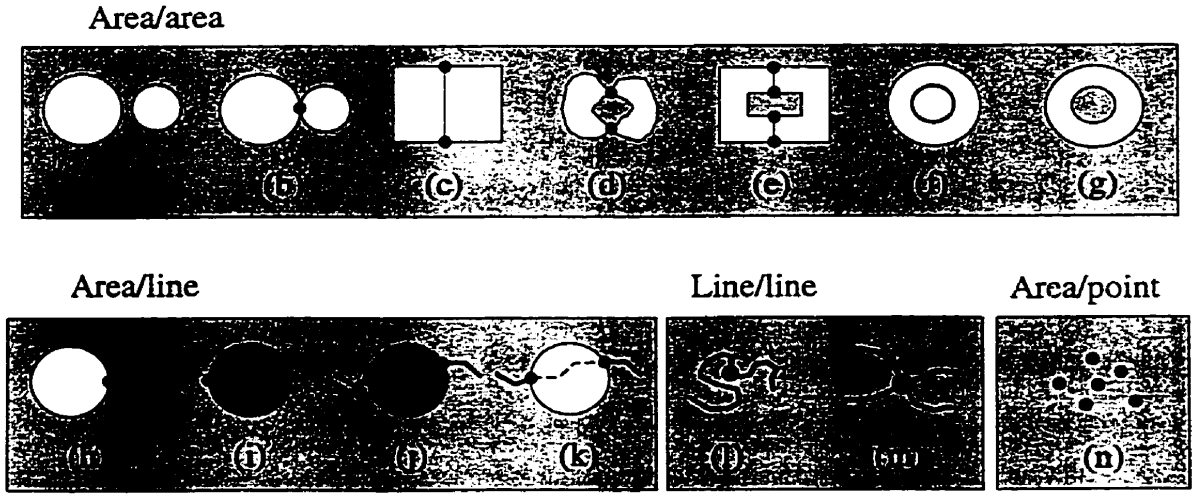


Figure 5.2 Topological relationships between spatial objects

5.4 The Voronoi Map Object (VMO) Class

The *VMO* class is an implementation of the geometric object classes and topological relationships presented in the preceding subsections. The implementation inherits directly the fact, resulting from the spatial object condensation technique, that the Voronoi diagram is embedded in the interior of a map object in M, and that the boundary of a VMO preserves the outline geometry of the map object in both connected subspaces. The following definitions would clarify the terminology to be used to define the VMO class. We note that this section is developed from a preliminary version, published in Yang and Gold [1996].

Definition 5.1: The *outline image* is the partitioning boundary left on the subspace S_1 from which the partitioning occurs. The *object outline* is the partitioning boundary duplicated on the partitioned subspace S_2 whose extent is confined in the enclosure of S_2.

Definition 5.2: An *object embedding* of a map in M implements the geometric properties of its composing objects in R^2. Except for that of the universal map, the object embedding is bounded by the object outline of the map and possibly outline images of other maps contained in the map.

For a single map, the object embedding disallows general intersection between objects, following the concept of the *single valued vector map* [Molenaar 1989]. This ensures the planarity of the map. The object embedding can be dynamic. That is, the components of the map object can be changed. However, any change to a bounded map object must be confined by the internal side of the object outline.

Definition 5.3: A *topological embedding* of a map in M implements topological relationships in R^2, over the object embedding of the map. Except for that of the universal map, the embedding is bounded by the object outline of a map and possibly outline images

of other maps contained in the map. The topological relationships are specified in the topological data structures.

The topological embedding is also dynamic, due to the dynamic property of the object embedding. The change of the topological structure depends on the dynamic object embedding and will also be confined by the internal side of the object outline.

Note that both object and topological embeddings of a map do not apply to the interiors of condensed objects in the map because the internal structures of these objects are suppressed. However, the outline image of a condensed object is topologically embedded in the map object as part of its object components.

Definition 5.4: The class *VMO* is a set of topologically and geometrically structured maps in M. Each instance of VMO has a topological embedding over an object embedding of the map instances. Denote $M = (P, LSG, L, PL, A, M_s)$, where P is the class of points, LSG the class of line segments, L the class of lines, PL the class of polylines, A the class of regions with closed boundaries, and M_s the bounded subclass of the class M which are geometrically presented as holes in M. The object and topological embeddings of a VMO are represented by the Voronoi diagram $V(M)$.

The definition is recursive. An instance of the VMO class can result in a hierarchy of maps. Nevertheless, a VMO may not have all classes of objects presented. The following implementations are valid for the object and topological embeddings of a VMO instance:

1) $V(M) = V(P, LSG, L, PL, A, M_s)$. This is the Voronoi diagram for a complete composition of spatial object classes. The existence of M_s class indicating that there are holes in the object embedding of the VMO.
2) $V(M) = V(P, LSG, L, PL, A, \varnothing) = V(P, LSG, L, PL, A)$. This is the Voronoi diagram of regions, polylines, lines, line segments and points.

3) $V(M) = V(P, LSG, L, PL, \varnothing, \varnothing) = V(P, LSG, L, PL)$. This is the Voronoi diagram of polylines, lines, line segments and points. There will be line networks but no closed area in the correspondent map.
4) $V(M) = V(P, L, \varnothing, \varnothing, \varnothing, \varnothing) = V(P, L)$. This is the Voronoi diagram of points and line segments. No more complicated objects are represented in the corespondent map.
5) $V(M) = V(P, \varnothing, \varnothing, \varnothing, \varnothing, \varnothing) = V(P)$. This is the point Voronoi diagram.
6) $V(M) = V(\varnothing, \varnothing, \varnothing, \varnothing, \varnothing, \varnothing) = V(\varnothing)$. This is the map with no components and is called a *null map* or *null object.*

It is understood that it is impossible to implement relatively more complex objects without first implementing primitive objects. The implementations 2) through 6) are Voronoi diagrams without holes, all spatial objects and their structures are presented at one flat level. The above definition has emphasized the spatial aspects of the VMO class. It is not, however, exclusive of any reasonable extensions to other data types to be included in the definition. For example, by extending some attribute types mapped to well defined attribute domains (e.g. stand_no => string, tree_type => string, density_rate => integer), an VMO object becomes a geo-object representing forest stands.

5.5 The VMO-Tree Organization of the VMO Class

Definition 5: A VMO-tree is a graph $G = (N, E)$ with node set N and edge set E. The components of any node x in N include a VMO object, denoted x.VMO. For any pair $<x, y> \in N$, an edge $e \in E$ exists between x and y iff the object embedding of x.VMO contains the outline image of y.VMO. The node x is called a parent, and the node y is a child. The VMO of the root node corresponds to the implementation of the universal map.

Each node in N is of the following data type:

(Node_ID, Parent_ID, Child_ID_List, VMO_ID)

where Node_ID and Parent_ID are unique identifiers of the node itself and its parent, Child_ID_List is a list of Node_IDs for the children immediately descendent from the node, and VMO_ID is the identifier of a properly defined VMO map.

An important design decision for the VMO-tree, as for a general distributed information system, is the implementation of object identity [Özsu and Valduriez 1991]. *Object identity* is the property of an object that distinguishes it from all other objects, which is independent of content, type, and addressability. It is the only property maintained across structural and behavioral modifications of an object. The objects manipulated by a general information system include *persistent objects*, the disk-resident objects concurrently shared by all users, and *transient objects*, the main memory-resident objects local to a program execution. There are two solutions for the implementation of object identity: the physical and the logical identifier approaches. The physical identifier approach equates the OID with the physical address of the corresponding object. For a database with a single server, implementation of the identity of persistent objects can generally differ from that of transient objects. Transient object identity can be implemented more efficiently with programming techniques such as using pointers. Problems with this approach occur when a distributed database is served and shared by different object managers. The reason for the problems is that physical addresses are not unique when boundaries of servers are crossed. The logical identifier approach, promoted by object-oriented programming, consists of allocating a system-wide unique OID (i.e. a surrogate) per object. The dilemma for managing object identification is a trade-off between generality and efficiency. The general support of the object model incurs a certain overhead.

We prefer the logical identifier approach. That is, every object managed by the VMO-tree must be uniquely identified by a logical identifier. This involves identifying nodes across the tree as well as spatial objects accommodated in each node. The identifier of a node can contain the following components:

- An integer number assigned by the tree constructor.

- The location of the metadata describing the node and its VMO. The description may include the owner of data in the node, the creation date and time, the object classes and services provided by the VMO, the authentic-check code for security of the map, etc..
- The host name serving the VMO data and its operations.

The VMO_ID may be composed of

- The name and location of the map.
- The customized interface ID.
- The object outline.

As far as spatial objects within a VMO are concerned, they are either persistent or transient. The persistent spatial objects created in a VMO are immediately integrated in both object and topological embeddings, which constitute the contents of the spatial database structure. The transient spatial objects are incurred mostly due to a spatial operation performed on a VMO. For example, another container VMO object may be created to hold a set of buffer zone boundaries. This VMO containing buffer zone is confined within the object outline of the host VMO. A transient object exist for the purpose of performing spatial analysis but may not be permanently preserved. It has the option to transform itself as persistent object if desired. In this case, the transient object is associated with the VMO from which it is derived and preserved as a *layer*.

There is one problem here. A spatial object wrapped in one VMO node does not have a unique identity in general. Since a VMO can be constructed within a host data management system which is independent of other hosts on the tree, the logical OID (integers) of spatial structures for two VMOs may well start from the same number and increment the value of OID as a new object is inserted. The OID generated by a local data management system is called the *local object identifier* (LOID) A LOID has to be globalized when the object it stands for is presented to other local object servers. This can be done by an OID module implemented into the base class of the VMO node. The OID module manages dynamic assignment of *global object identifiers* (GOID) to node objects on the tree, and mapping

between LOID and GOID [Schmitt and Saake 1995]. The working principle of the OID manager is based on a uniqueness scope model (Figure 5.3) [Kent 1991] which identifies objects across a federated (maybe heterogeneous) database management system. It turns out that the scope structure has a direct application to object identification accommodated by the VMO-tree.

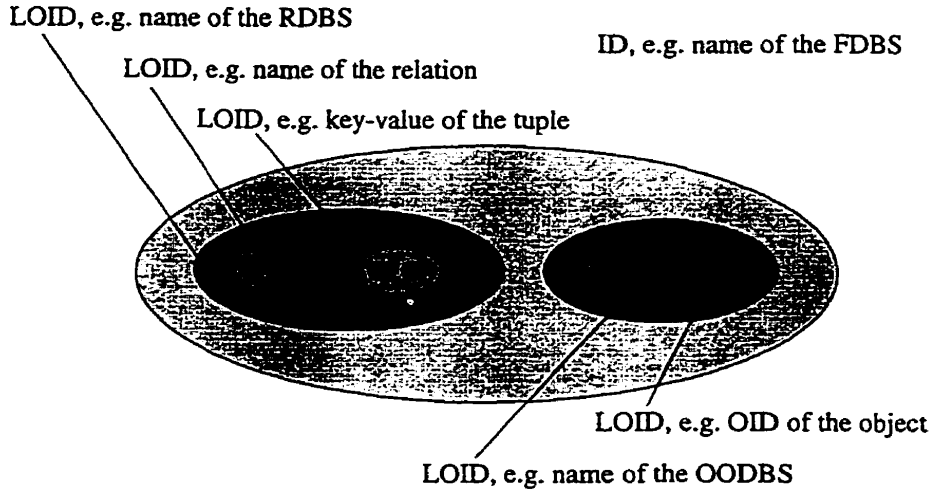


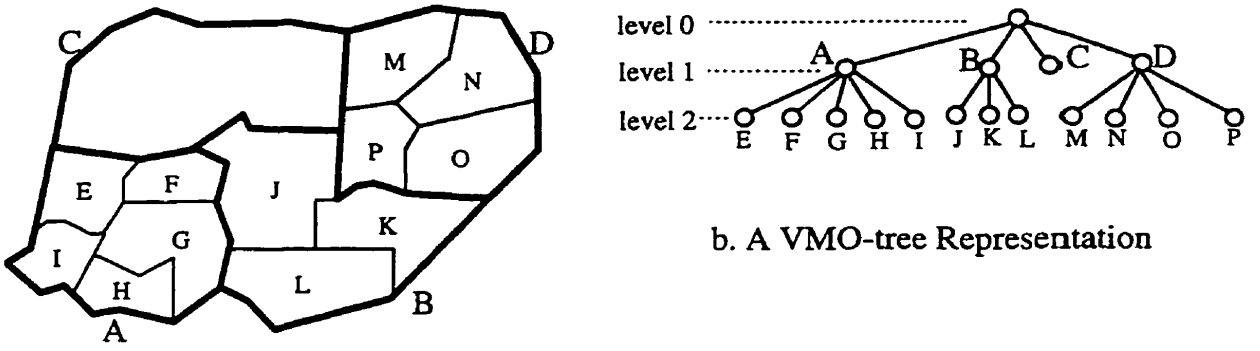
Figure 5.3 The scope model (adapted from [Kent 1991])

5.6 The Construction of the VMO-Tree

The construction of a VMO-tree can be either top-down, bottom-up, or a combination of both [Yang and Gold 1996]. There are two scenarios when the top-down strategy is used. In either case, the universal map of the root node is first created. One scenario of the top-down approach condenses a set of map objects for each smaller geographical area such that the whole study region is completely partitioned by the outline images of these null objects. The nodes corresponding to these map objects are added into the VMO-tree at level 1. Each map object is assigned to a host whose location is known to the root node. Before the assignment, the server of the root node negotiates with the child-server for the proper location of the map. Once the agreement is reached, the data about the VMO child is

transmitted to the child-server. The child-server can then load the map in memory and work on the detail of the map. Even smaller map objects can be added in the tree at lower levels (Figure 5.4).

Another scenario of the top-down construction is that the root server works on the detail with the universal map and adds components in it. Some heuristic approach can be taken to aid the decision on when and where a cluster of components can be condensed. Suggestions on the heuristic criteria include using a threshold for the total size of the map, localized thresholds for the density of spatial objects in heterogeneous clusters, or some artificially implemented triggering methods. Whenever a new object is dynamically condensed, the node representing the object is inserted into the tree. Meanwhile, the server decides, in consulting with the database administrator, where the condensed map should be stored.



a. A Partition of a Region

b. A VMO-tree Representation

Figure 5.4 The top-down partition and its VMO-tree (after Yang and Gold [1996])

The bottom-up strategy works in a reverse way where the smallest distinct objects are first composed as maps separately. The construction assumes the existence of the root and inserts into the universal map all the object nodes corresponding to the maps created previously. This involves pasting Voronoi diagrams into the universal map. All the pasted objects will be at level 1. This results in a flat tree where all children have the root as the parent. A generalization process can then take place which aggregates smaller objects into bigger ones. When a bigger object m aggregated from k smaller objects at level i is formed, the object m becomes the parent of the k objects. Note that the aggregation happens at level

i - 1. The new node representing object m will therefore be at level i and the k children are dropped to level i + 1. The list of the children in the original parent at level i - 1 will be updated, which excludes the k objects and includes the new aggregated object m. Likewise, the Parent_ID in each of the k objects needs to be modified. After the generalization, the tree becomes narrower by k - 1 branches at level I, and deeper if the generalization is applied to a number of leaf nodes. The generation process can be performed heuristically, with predefined rules and possibly interactive instructions from the database administrator.

It is also possible that intermediate nodes are generalized without the knowledge of the root. These intermediate nodes serve temporarily as roots of subtrees. A forest can exist during the whole process. A bigger tree is federated by inserting a number of roots of smaller trees.

Both the top-down and the bottom-up approaches may be interchangeably applied during the construction of a distributed spatial database, which either partitions a map of an intermediate node, or aggregates smaller maps.

From a cartographer's point of view, the top-down approach works from smaller scale, larger sized objects towards larger scale, smaller objects. The deeper down in the VMO-tree, the more details an object can expose. On the contrary, the bottom-up approach works from larger scale, smaller sized objects towards smaller scale, larger objects. The higher a node is in the VMO-tree, the more abstract it becomes.

Inserting an existing condensed object into the VMO-tree, or aggregating smaller condensed objects into a bigger one, is a compound operation which takes several steps. The principal technique for managing the geometry and topology has been described in Chapter 4. The realization of the technique depends on intelligent decisions on choosing the boundary of a partition or an aggregation. It also relies on effective communication through the computer network if different VMO servers connected to the network are distributed at separated geographical sites.

It is without doubt that building a distributed or federated spatial database for a region or an enterprise based on the VMO data model is not just a technical issue, but more an endeavor which requires institutional, societal, and cultural change. The data model makes the technology available. The adoption of the technology for a region-wide spatial DSS, however, needs the commitment of policy makers, and managers in the public and private sectors at strategic, operational, and user levels.

It is noted here that at the first glance the VMO-tree looks similar to some of the other trees such as the R-tree or Cell-tree. They are essentially different in two major aspects:

Firstly, the R-tree or Cell-tree like structures are geometric ones. The partitioning of the geometric objects is the primary concern when those trees are constructed. The VMO tree is not only a geometric, but also a topological structure. The reason is that within each partitioned cell of the VMO tree there implemented with an topological structure. Because of the topological representation, the VMO tree is ready to answer topological queries and to perform spatial analysis. While the objects in the cells of other geometric trees are not topologically organized (spatially-unaware linked list is one common data structure), they are not ready to answer topological queries or to perform spatial analysis. This is the very reason to categorize them as geometric structures.

Secondly, the partition criteria and mechanism of the VMO tree is significantly different from those of the geometric trees. In each subspace to be partitioned, the topological construction and functions with the VMO tree makes the selection of partitioning boundary "intelligent" in that it is topologically, thematically, and semantically conscious. As a result, the partitioning is not restricted by a rigid shape or size and produces meaningful objects.

5.7 Constraints of the VMO Model

The VMO model is composed of a set of properly defined VMO objects which may be physically distributed and are managed by the VMO-tree. The VMO-tree represents a more abstract topological structure which is mapped from the partition of a Euclidean space structured with a neighbourhood relationship. Hence the VMO-tree is a directory of a repository of integrated maps which may be operated upon by different database servers. Each server can update the directory below its node, which makes the directory dynamic. In order to maintain the integrity of the whole database, constraints have to be applied to each node of the tree. Whenever the directory is updated, the checking of the constraints needs to be triggered.

In general database theory, constraints operate in parallel with the structures of a database schema. They can be used as guidelines for deciding the schema's structure according to three principles: representation, nonredundency, and separation [Beeri et al. 1978]. The *representation* of a constraint needs to include the natural relationships between objects. The constraint can be thought of as a property of the schema which should be true with respect to all such natural relationships represented in the schema. *Nonredundency* is a property of constraints which emphasizes that a constraint, derived from the structures and other constraints already specified in a schema, should not be redundently specified. It may be confusing to represent the same information in more than one way. *Separation* is devoted to a schema in such a way that information units, as represented by constraints, are separated and do not interfere with one other.

Based on the above guidelines, we define the following constraints for the VMO-tree structure. Let $A = \{a_1, a_2, ..., a_n\}$ be a finite set of condensed objects contained in a map object. A is the parent and a_i (i = 1, 2, ...n) are children. These constraints state:

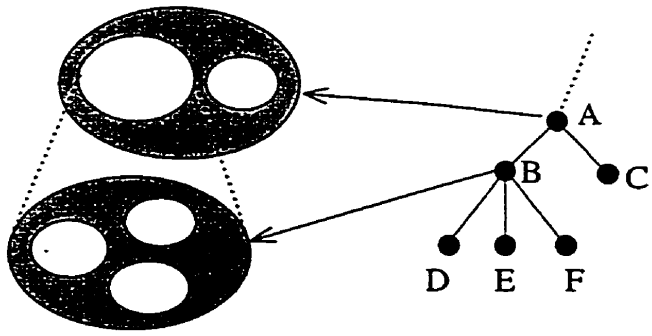
Constraint 1. Every node of a VMO-tree, except for the root node, has one and only one parent. The root node corresponds to the universal map which is unbounded.

Constraint 2: The number of condensed objects contained in one map object corresponds to the number of children under the node representing the map object. Adding a child node under a parent node adds a condensed object in the map corresponding to the parent node. Deleting a node from the tree deletes the corresponding condensed object in the parent map and all child nodes descendent from it.

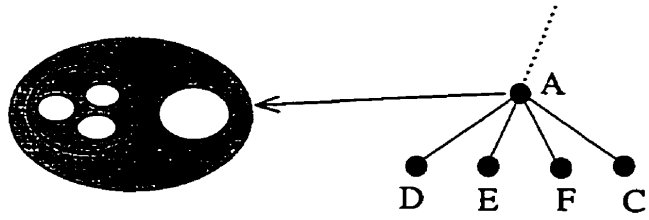
Technically, the deletion of a node can be achieved by pasting the corresponding condensed map back to its parent map and making the map a region. In this case, all the immediately descendent nodes of the source node need to be promoted one level higher and to have the target node as their parent (Figure 5.5).

Constraint 3: For $a_i, a_j \in A$, $(i \neq j)$, $a_i^\circ \cap a_j^\circ = \varnothing$. That is, interior spaces of any two distinct condensed child objects must be disjoint at the same level. Overlapped or partially overlapped subspaces enclosed by outline images are disallowed. The non-empty intersection of distinct member of A may only happens on the outline images.

Constraint 4: $\bigcup_{a \in A} \text{outline}(a) \subset \text{outline}(A)$. That is, the union of outline images of all child maps is a subset of the outline of the parent map. The complete outline set of A is the union of the object outline of A and $\bigcup_{a \in A} \text{outline}(a)$.



(a) A partial VMO-tree



(b) The partial VMO-tree after node B merged into node A

Figure 5.5 A partial VMO-tree before and after deleting a node

5.8 Operations on the VMO Model

The VMO model is composed of a set of geometric object classes which are topologically implemented with the VMO class. The integral relationship between VMO objects and the other spatial objects is represented by the VMO tree. The VMO tree also serves as the directory of the federated spatial database.

Based on this understanding, operations on the VMO model can be categorized as the ones on the VMO tree and the ones on the spatial objects addressed by each node of the tree. We have just seen the construction operations and constraints on the VMO tree in the previous sections. We have also discussed adding, deleting, and moving primitive spatial objects, as well as typical GIS operations with Voronoi diagrams, in Chapter 3. What needs to be explored in this section are additional operations and constraints particular to the introduction of the map object class and the tree structure. These include set-theoretic

operations, spatial queries and their optimizations, and operations related to multiuser database function and network communications.

The set-theoretic operations typically involve two object sets which can have either identical or different attribute types or themes. Formal and detailed discussions about the set-theoretic operations on sets of areal objects can be found in Worboys [1992a]. These include calculating unions, intersections, and complements, with respect to given themes and objects. New instances of the VMO-tree can be generated by a set operation. An example of this calculates the intersection of two objects, each from a VMO-tree instance. A third instance of the VMO-tree can result if the calculation results in a non-empty subset of objects.

A special care must be taken when any of the two VMO objects contains maps, i.e. holes. Figure 5.6 illustrates a binary operation involving VMO objects *A* and *B*, while *B* contains a map object presented as a hole *C*. The binary operation has to be performed first on *A* and *B* and then on *A* and *C*. The final result will be the union of the two-step calculations. It is possible to repeat this process when *C* again contains holes.

Figure 5.6 Binary operations involving objects containing holes

Spatial queries can be greatly facilitated by the VMO-tree, which provides a fast search along the edges of the tree, and the topological embedding within each spatial object, which enables navigation and searches over components of a map object. This ability to search objects vertically and horizontally resembles that of the B^+ tree.

Both set-theoretic and spatial search operations involving multiple VMO objects can be distributed and executed in parallel. That is, instead of sequentially performing an operation at the entry level objects and then preceding to lower level ones with a single processor, a

quick determination of any lower objects involved should be fired with the help of the node structure. The operation can then be divided and executed concurrently on all sites or processors hosting these objects. Certain optimizing criteria need to be considered concerning the efficiency of this feature. This may involve factors such as the cost and complexity of the operation, the size of the object set, and the cost-effect comparison on data transmissions through the network.

The VMO-tree manages map objects, some of which may reside in or be accessed from remote sites. The object networking functions must ensure the data security, integrity and concurrency controls, and dynamic communications. The integration of database systems technology and a network environment leads to a new class of problems and consequently to new means to solve them. These problems and solutions are active research topics of computer science. An important aspect of the problem is the need for network-wide definition (or directory) of the location and characteristics of data objects. This definition should include the details about how information is partitioned and if it is replicated on different nodes of a network. When a user fires a query access to data, the local node must determine where the data is located, if the data is on a remote mode, and if the user has permission to access the data.

A client-server architecture can be designed to allow node objects of the VMO-tree to be linked to client applications. The client-server architecture is based on the concept of distributed processing, with the front end (or the user application) being the client and the back end (the database access and manipulation), the server. This facilitates setting up enterprise-wide connectivity. The functions performed by the server include:

- Centralized data management
- Data integrity and database consistency
- Database security
- Concurrent operations (for multiuser access)
- Centralized processing (e.g. stored procedures precompiled and stored in a database -> encapsulated functions akin to objects)

The client is responsible for handling user-specific database access tasks, and the server for managing shared data. The functions of a client:

- Customized user interface
- Front-end processing data
- Initiation of the server remote procedure calls
- Access to a database server across a network

A data dictionary plays an important role in applications using the client-server architecture. A data dictionary is a manual or automatic repository, usually managed by the database, containing information about applications, databases, logical data models and constituent objects, users and user access authorizations, and any other information useful for defining the organization and use of data within the database. It provides location transparency for the user/developer in viewing data dispersed around the network. The contents of an object data dictionary contains the following key components:

- Class definitions
- Class hierarchy
- Object access authorizations
- Indexes
- Server definitions (and locations)

An issue of interest is that the dictionary itself must be distributed; either there are identical copies on each server or each server maintains a data dictionary that describes components on that server only. We hope these issues will be explored more deeply concerning the client-server architecture with the VMO tree, as research and development continues to progress. The definition and organization of the VMO class with a clear boundary seems to be a promising way of implementing distributed client-server applications.

Chapter 6

The Design of the VMO Forestry Data Management System

We propose, in this chapter, the design of a forestry data management system based on the VMO model to handle spatial properties of the real world. For simplicity, we name the software system FORMONET, meaning FORestry Map Object NETwork. The design process is guided by an object-oriented methodology, called the *Object Modelling Technique* (OMT) [Rumbaugh et al. 1991]. We note here that the result of the design in this chapter will not be a complete application system – more issues other than technological ones must be addressed and resolved for that. Instead, the design is aimed at an outline of the spatial data management component, especially the linkage between system components and the VMO class. Three closely inter-related models (object, dynamic, and function) will be sketched. Useful methods of major classes will be identified and described, with the purpose of demonstrating the roles of object classes in the system. In the detail design and implementation phases, a geographical entity must be associated with a spatial object in order to complete the database of the system. Given the VMO model, most geographical entities can be associated with types of spatial objects. The design can be used for understanding and communication between software requesters and developers, and as a framework for detail design and implementation.

While the design in this chapter is concentrated on the system architecture in which the VMO spatial object model is used, other works conducted in the Centre for Research in Geomatics at Laval University have identified more specifically forestry entities and relationships. Doucet [1990] described the development of a spatially referenced information prototype system for forestry data management. The design in this project is currently evolving into an operational system, named Système d'Aménagement Forestier Informatisé à Référence Spatiale (SAFIRS) (cf. Szarmes [1997]). Bédard et al. [1993] proposed a prototype of spatio-temporal forestry database where a forestry data dictionary is

compiled. Theoretical solutions to the management of changes in forestry objects, using a commercial GIS (MGE-Dynamo), were discussed (Bédard [1993]). Felten [1998] reported an experiment under the MGE environment to integrate time into forestry maps, based on real forestry data from Montmorency Forest which is located at the north of Quebec city. It would be beneficial to entail the attributes of spatial object classes proposed in this chapter, with the descriptions about forestry seen in these previous results.

6.1 Problem Statement

Following the guideline of the OMT method, the first analysis task is to formulate the problem statement. The statement speculates on what needs to be done, not how this can be done. Based on the discussion in Chapter 1, the forestry domain knowledge, and current practice in the forestry industry and in government departments, the statement for developing a software system FORMONET can be documented as follows:

> FORMONET is a computerized system to support forestry data modelling and management. The system must be maintained through a dedicated network of hardware, software, and cultural components (ref. Ch. 1). There are three kinds of management mandates which involve managers at the strategic, managerial, and operational levels corresponding to geographical entities at national, regional, and territorial geographical scales. The fully functional system needs to support the gathering and analyzing of forestry domain information from the national office in order to facilitate policy-making and to support strategic planning concerning sustainable forestry. The regional offices in turn prepare timely forestry information which is updated and which bears on natural and man-made processes occurring throughout the territory. A territory is composed of forest stands, which are the atomic units for harvesting, planting, and silvicultural operations.

The core of the system must store and manage digital maps at various scales over a set of geographically diverse sites. The sources of data include existing paper maps, airborne photographs, remotely sensed images, GPS and other field survey records, and any documents describing or relevant to these data. All data must be digitized and stored in the geo-database of the system (allowing different formats). Data initialization can be done at authorized sites. These sites are consequently owners and hosts of the distributed data within a federated database. The host is responsible for maintaining, updating, and accessing information upon request. The practice of quality checking and calibration for the data on each site must be performed as a result of agreement between responsible members of the system. For spatial objects contained in a map, an integrated topology and geometry needs to be maintained at progressively larger scales for smaller regions. Aspatial data is associated with spatial data. In the case that the detailed data of a particular site is temporally unavailable, the outline shape and general information about the corresponding spatial region needs to be present at a higher level.

The system needs to support different types of accesses to the database for system developers, application programmers, and occasional users through various interfaces. The access to the information system should be controlled through the assignment of privileges to use the system. A dedicated graphics user interface should be provided for each public outlet of the system. The interface accepts and launches user queries, analysis, and simulations, and presents results returned by the system. Duplicated storage of data is not encouraged, for the sake of data consistency. Some users can have a copy of a part of the data retrieved from the system for research and analysis. Whether the modified information of the copy needs to be incorporated into the system is the decision of managers at appropriate levels of the system.

In the following, we include an example concerning data resources and user required services when considering setting up a Forestry Geographic Information System [Lamont 1988] in the province of Manitoba, Canada. Conditions may vary from project to project and site to site. The example illustrated some rather common features of a forestry information system in real world.

The input information of the system comes from the following maps:

- Forest Resource Inventory Maps
- Forest Wildlife Maps
- Natural Disaster Maps
  - Wind, drought, flood
- Forest Insect and Disease Maps
- Silviculture Activity Maps
- Timber Management Activity Maps
- Soil Maps
- Land and Resource Use Maps
- Utility Corridor Maps
  - Proposed and/or actual
- Forest Capability Canada Land Inventory Map
- Satellite Imagery
  - Landsat and variations

User required services from the system (partial):

- Data Manipulation
  - Analysis of effect of management decisions
  - Area analysis
  - Buffer zone analysis
  - Wildlife loss analysis
  - Overlay analysis (125,000 polygons at a time)

- Updating
  - Review map and attribute files
  - Review status and ownership boundaries
  - Enter new features as they are constructed
  - Wildlife area and loss calculation and revision of original forest cover data
- Thematic Mapping
  - Colour code and shading code options to identify selected forest stand attributes
  - Determine a new colour or shading code based on results of polygon overlay
- Site Specific Information
  - Integration of town street maps into the database where specific items at large scale (1:500 - 1:2,000) are to be positioned
- System Functions
  - Polygonization - areas connected to form polygons
  - Merging and dropping lines
  - Subdivision of polygons
  - Browsing
  - Define corridors along straight or convoluted lines
  - Complex polygons
  - Windowing
  - Scale changes - full range performed in computers
  - Projections - to increasing graphic capabilities for large-area mapping

The example shows that forestry data may come from a variety of sources and disciplines, with differing formats, types, and scales. The services of a forestry GIS also falls in a large range, from manipulating individual forestry objects to operations on whole maps with large volumes of polygons. Some of the required services are dynamic, such as merging and dropping lines and subdividing polygons, forming polygons from connecting areas, and inserting new features as they become exist. There are also some technically challenging services, such as changing the scales which requires automated mapping generalization capabilities, and revising original forest data coverage which requires temporal modelling,

manipulation, and queries. The requirements shown in this real world example demonstrate a close correspondence to the analysis presented in Chapter 1.

6.2 The Object Model

Based on the problem statement, major object classes are identified and their relationships are presented as the object model shown in Figure 6.1. The diagram uses the OMT connotations to denote object classes and their associations. The class USER represents a collection of users of the system, which can be managers, developers, decision makers, or casual information seekers. GEO-DATABASE class is the data store, which contains relevant geo-objects according to a forestry inventory. USERS can access GEO-DATABASE through interfaces of the data store. Closely related to the geo-object database is the WORKSPACE class. The objects in this class constitute part of the geo-objects in the geo-object store. The spatial objects loaded in WORKSPACE are implemented as VMO objects. FORESTRY PROJECT is the class that contains tools for users to specify, set up, and to manage and manipulate an application project. Besides accessing geo-objects from the geo-database, located through WORKSPACE, a user works on a forestry project can also collect and integrate any related documents and multimedia information for the project.

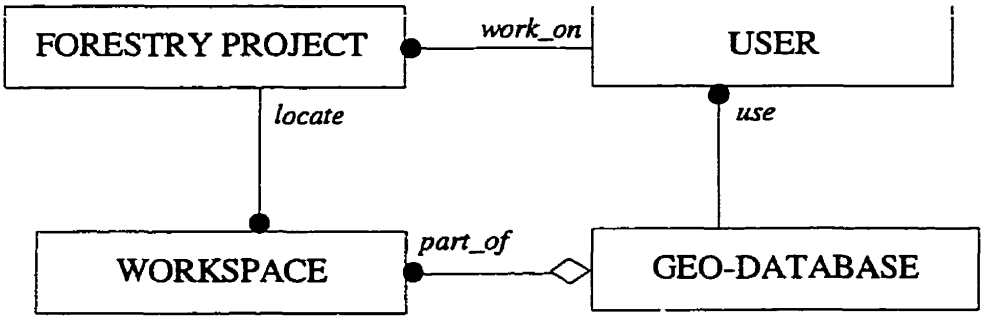


Figure 6.1 The object model for FORMONET system

For the USER class, attributes may include name, address, and account information which properly identifies the role and access privilege of a user and the nature of using the systems. When integrated with a corporation information system, the connection between the USER class and the corporate database would have to be established.

The GEO-DATABASE itself forms a subsystem and provides rather independent functions. The subsystem can be decomposed into a few functional components, such as a database modelling tool, a component taking care of storing and accessing geo-objects, and a graphic user interface. The database modelling tool helps users to conceptualise and classify geo-entities and their relationships. A good example of such a modelling facility is Modul-R, a CASE tool based on an E/R model and used to formalise geo-spatial phenomena in a database and to generate data dictionary for it [Caron 1991, Caron and Bédard 1993]. The geo-object storing and accessing component can be a commercial database such as Access, or Oracle. The database schema comes from the result of the modelling tool. It is noted that some relational databases nowadays are extended to accommodate spatial objects. An example is the Spatial Cartridge, employed by Oracle8™. Figure 6.2 illustrates partial forestry geo-objects for the GEO-DATABASE. Note that each geo-object class has in its attribute list an Id of the corresponding VMO object.

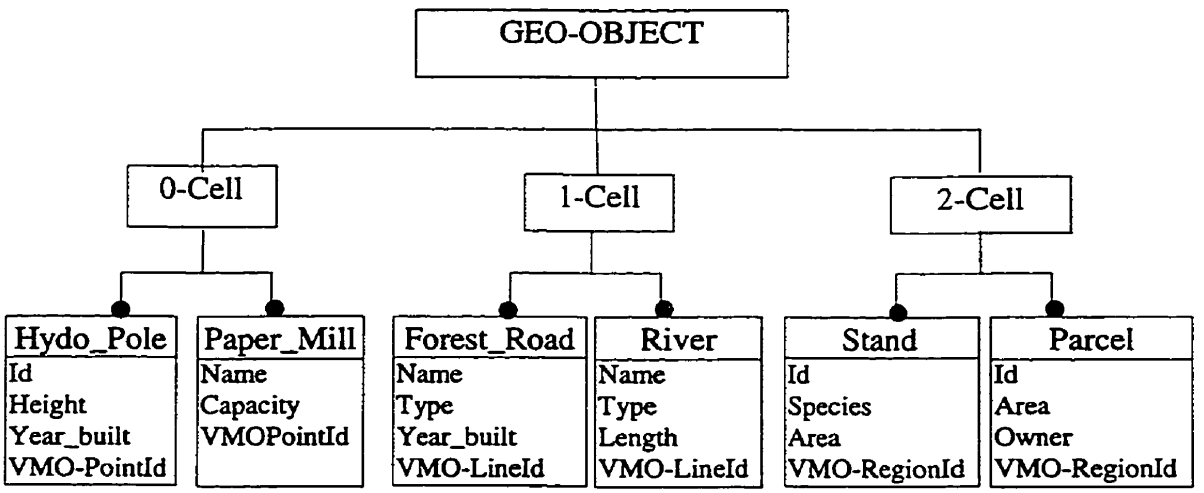


Figure 6.2 Partial Forestry Geo-Objects

FORESTRY PROJECT is the component that helps users to set up, manage, and operate on forestry projects. Project name, purpose, developers, and history are among important attributes associated with the class. Included with the class are metadata describing locations and nature of data resources and auxiliary documents related to a project. QUERY, ANALYSIS, and SIMULATION are functional components of the class, which work through the WORKSPACE class. For each of these functions, some templates can be constructed through which instruction statements from users will be accepted. The functions will then validate and optimize the statements and parameters. Corresponding actions on the WORKSPACE and GEO-DATABASE will be triggered. Results are sent back to FORESTRY PROJECT and presented via users desired reporting templates. One of the templates for QUERY could be a dialog box for composing SQL statements. Other templates featuring pictorial languages can also be developed.

The class WORKSPACE is the central concern of this chapter, and is directly related to the VMO data model developed in this thesis. The primary role of the class provides users visual platform and tools to design, edit, manipulate, query and analyse spatially referenced geo-objects. The visual platform contains a graphic window displaying those interested geo-objects. The graphic representation of geo-objects is called a map in common sense. An instance of the class is the workspace specifically designed for an instance of FORESTRY PROJECT. A workspace may involve one or many maps covering different geographical areas or themes. Most important attributes of the WORKSPACE class include the name of a workspace, the number, names, and locations of maps, and the layout or settings of maps preserved from the last login.

The class WORKSPACE can be directly associated with the VMO Node class developed in Chapters 4 and 5. The dependency and the spatial aspect of the VMO object model is shown in Figure 6.3. Note that the arrow from Geometry to Topology represents a dynamic process to construct topology from geometric objects. Dynamic modelling will be discussed in the next section.

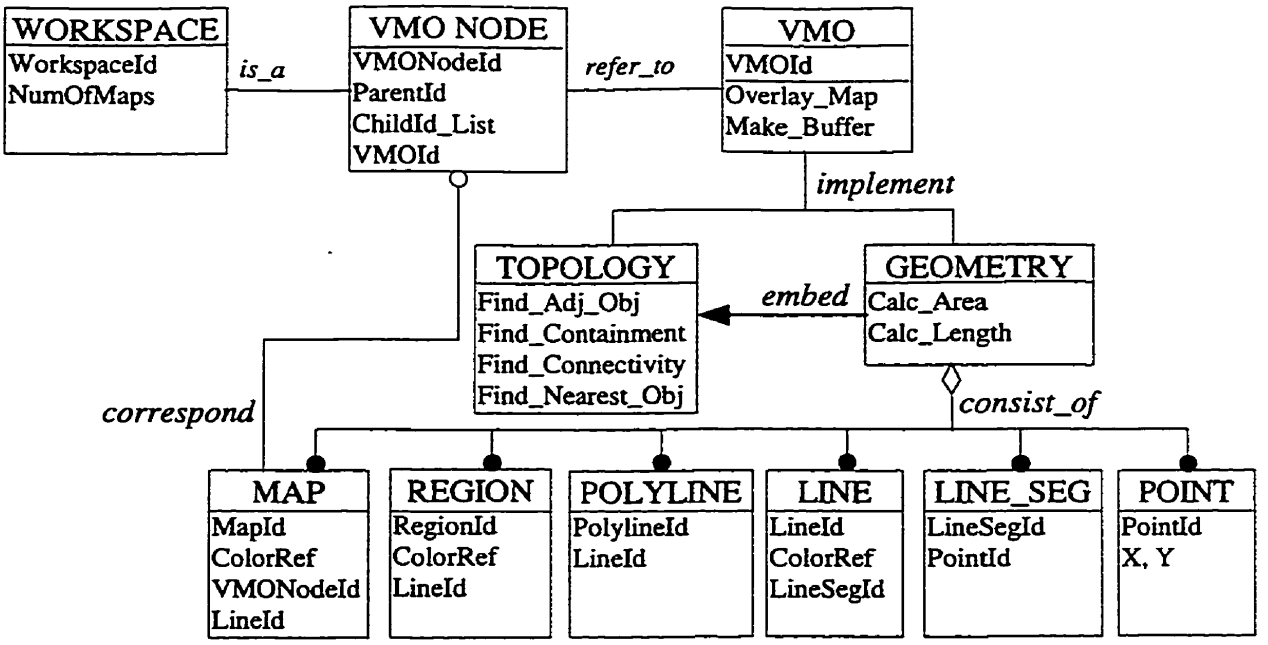


Figure 6.3 Dependency between WORKSPACE and the VMO model

In Figure 6.3, the WORKSPACE is represented as a specialised VMO NODE which further refers to VMO class. One specialisation adds graphic object manipulation tools in the visual platform of WORKSPACE. A VMO object is implemented with geometry and topology, which enables the VMO object to provide topological queries and spatial analysis. The VMO presents a seamless accessibility to geo-objects, whether they come from a single or multiple geo-databases. Recent software technology such as COM (Common Object Model), DCOM (Distributed COM), and OLE DB for OLAP (Online Analytical Processing) ensures this capability.

6.3 The Dynamic Model

The *dynamic model* shows the time-dependent behaviour of the system and the objects in it [Rumbaugh et al. 1991, pp. 169]. An interactive system or object acts upon events. A contemporary GIS necessarily involves human-computer interaction. In this context, any actions generated by users and devices send messages to the system which are interpreted as

events. The objectives of the dynamic modelling are therefore to find out possible events associated with the components of the system and to design proper responses for them. Note that some responses from a component may incur events to other inter-related components within the system. The flow of events is a *scenario* which describes things happening between objects as a result of external stimuli [Rumbaugh et al. 1991, pp. 173].

For a complex system such as FORMONET, there are many scenarios for various application tasks. These include tasks for constructing and populating a geo-database, for preparing and performing spatial analysis, the result of which may or may not become permanent objects in the database, for various queries which require searching through the database and which nevertheless do not modify the status of the database, etc. It should be noted that most of these tasks generate events requiring responses from the geo-database management system and that the fulfilment of the triggered tasks relies on the internal processes enabled by the database system. The internal processes interact with objects in the database and are transparent to the user. Therefore, the interactions at the surface level between the system and user involve only a few object classes. Based on the object model developed in the previous section, these objects typically include USER, FORESTRY PROJECT, WORKSPACE, and GEO-DATABASE. This implies that a general scenario involving these objects can be generated (Figure 6.4).

Figure 6.4 illustrates an event flow diagram from USER to GEO-DATABASE, after a FORMONET GUI interface is created. The classes Window and Message loop are objects provided by the operating system under which the application is executed. They are included in the diagram to show the actual process by which application events are generated. A USER gives instructions through manipulating system devices on the GUI interface, which generates events. These events are first received by the operating system which then sends messages to a Message loop associated with the application instance. Messages are then taken one at a time (for a single processor application) and dispatched to the message handler embedded within the GUI interface. Window class objects automatically handle a set of predefined messages whose execution is taken care of by

methods implemented within a Window object. Customized messages have to be handled by an Application whose major components are shown within the dashed-line rectangle. In the diagram, it is shown that FORESTRY PROJECT receives customized messages and generates events to occur within a WORKSPACE. The WORKSPACE further invokes a VMO-server which could be bounded by the WORKSPACE. The VMO-server takes care of connectivity issues and parses events with the DDL or the DML database languages to be developed with the spatial database management system.

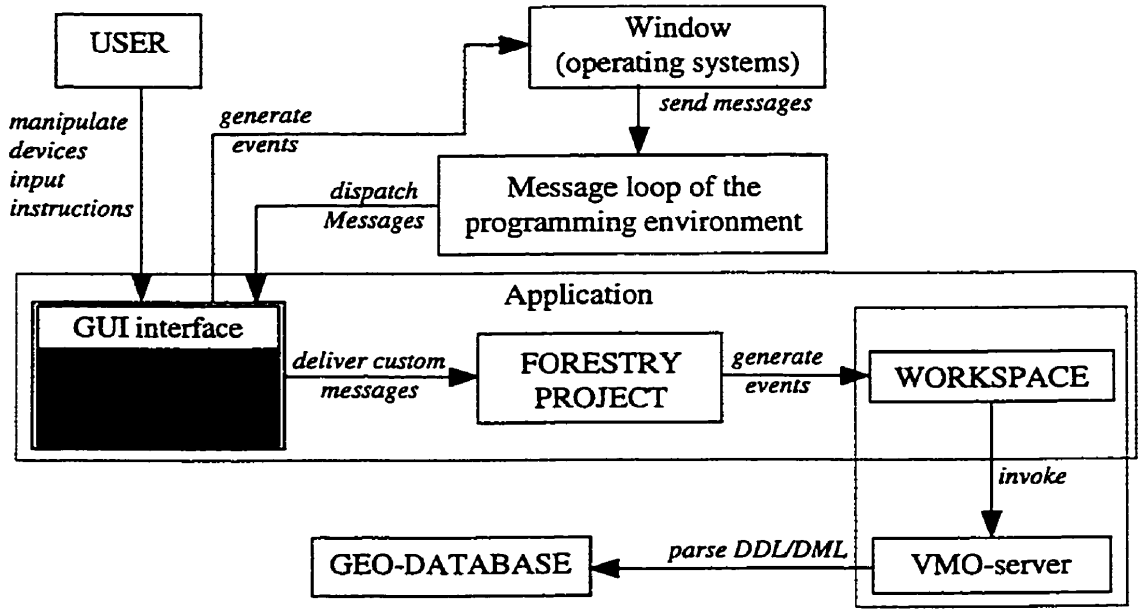


Figure 6.4 A general event flow diagram for FORMONET

The general scenario for the sequence of events is by no means complete. For example, for an event sender, an immediate response from the event receiver (occasionally the sender itself) is required before any subsequent event is sent. The event flow diagram nevertheless demonstrates the main path central to the system.

More detailed design of the dynamic model should include an event trace diagram for each particular scenario, in which the time-dependent order of events between objects is scheduled according to results of a preceding event. State diagrams for each object class should then be worked out which entail the dispatch of events and the control of an event flow by a state with more than one exit transition. A *state diagram* shows the dynamic

behaviour of an object at a particular state. When an event occurs, the next state of an object depends on the current state as well as the event; a change of state caused by an event is a transition. Therefore a state diagram is a graph whose nodes are states and whose edges are transitions labelled by event names [Rumbaugh et al. 1991, pp. 173-179]. The OMT technique provides various basic and advanced constructs for the design, generalization, and control of event flow and state diagrams. We will elaborate only one state diagram as an example and leave the rest for future development and implementation.

We note here that customized dynamic events can correspond to evolution of an object. Catching, processing, and saving these events contribute to the temporality of the database. It has been discussed in Chapter 3 that the underlying topological and geometric operations of the VMO model are associated with a log file which preserves the history of a map at the scale of points and line segments. We illustrate here how a node of a dynamic model can be refined into a state diagram within a workspace, which handles events for the creation, update, and destruction of higher level objects such as a region or a map.

Figure 6.5 shows a scenario for an interactive operation "close polygon" and the corresponding message passing section in a high-level dynamic model. The event will be handled by the VMO-server associated with the WORKSPACE. The state diagram for VMO-server may be modelled as that shown in Figure 6.6 which is a simplified version of the actual implementation.

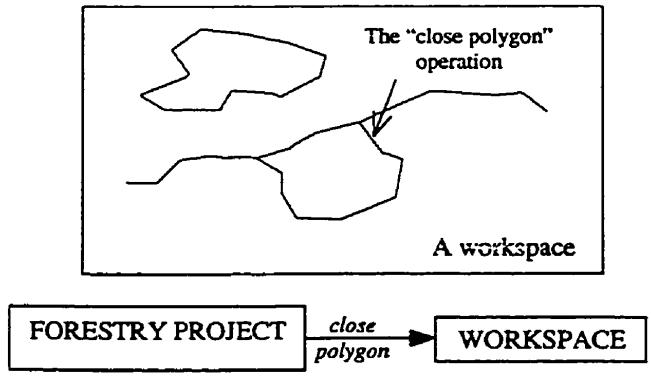


Figure 6.5 A "close polygon" dynamic event and a section of the dynamic model

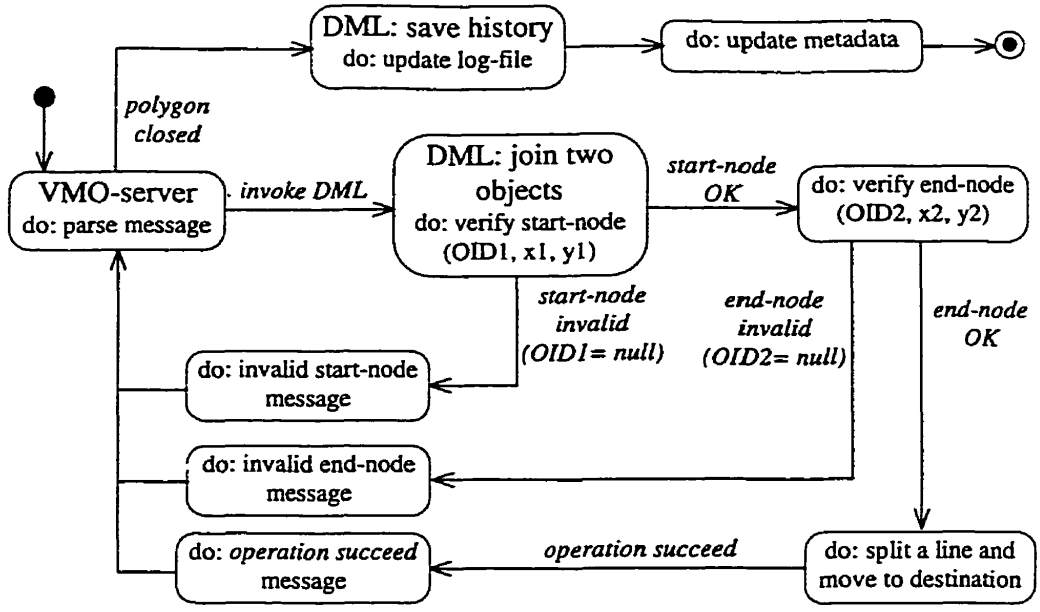


Figure 6.6 The state diagram for the VMO-server in the WORKSPACE to handle the "close polygon" event

6.4 The Functional Model

A *functional model* describes the computations within a system, through the use of multiple *data flow diagrams* [Rumbaugh et al. 1991, pp. 124]. A data flow diagram is a graph consisting of processes as nodes, and data flows as edges. It also attaches actor objects that produce and consume data, and data store objects that store data passively. Each data flow diagram shows the flow of values from external inputs, through operations and internal data stores, to external outputs. A data flow diagram does not generally show control information, such as the time at which processes are executed or decisions among alternative data paths; this information belongs to the state diagrams in the dynamic model. A data flow diagram does not show the organization of values into objects; this information belongs to the object model [Rumbaugh et al. 1991, pp. 124, 180-181].

A *process* transforms data values. It may be a high-level functional tool set or an indivisible function. The lowest-level processes are pure functions without *side effects* [Rumbaugh et al. 1991, pp. 124-132]. Typical pure functions include the sum of two numbers, the recording of a transaction into a log, and the drawing of a line through a list of points. A complex system needs to generalize really high- level processes to make data transformations clear. An entire data flow graph is a high-level process. Low-level processes need to be worked out as the design moves more towards implementation. For example, calculating the tree volume for a territory is not a pure functional process. It involves searching individual stands covered in the territory and calculating areas for the stands. A process may have side effects if it contains non-functional components, such as data stores or external objects. "The functional model does not uniquely specify the results of a process with side effects. It only indicates the possible functional paths without showing which path will actually occur" [Rumbaugh et al. 1991, pp. 125].

As with the dynamic model, we provide in this section a general functional model for FORMONET (Figure 6.7). The idea of the functional model is to demonstrate data flows transformed through major processes for the system. It is assumed that a digital map of the working area, in addition to other interface templates, is required. The digital map displays itself on a graphics window as users interact with the system to achieve goals.

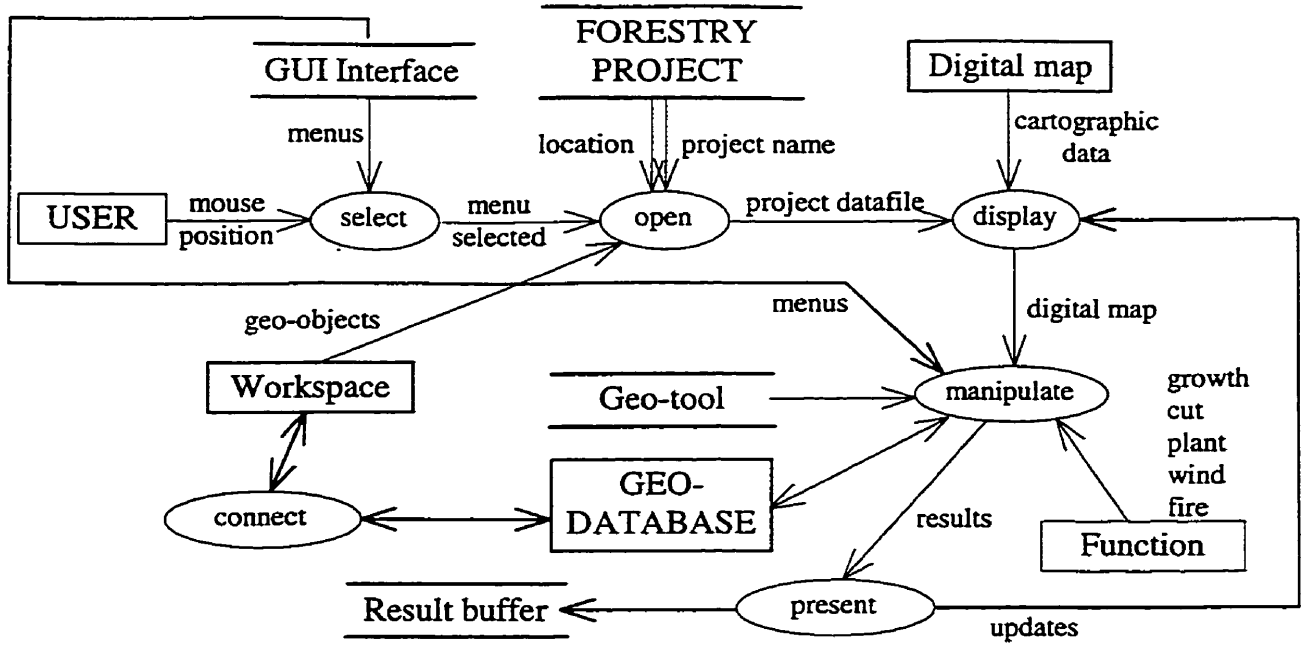


Figure 6.7 The functional model for FORMONET

Major processes shown in the diagram include the usual functionality of the system. "Select" allows users to choose a project to work with. Menus are provided by the GUI interface. The inputs of the process are mouse positions (or information from using other input devices). The output of the process is a particular menu selected. The "Open" process lists all project files from a data store from which users pick up one project to open. This process also needs to locate a workspace and geo-objects contained in a project and possibly to connect a geo-database to use other geo-servers. "Open" then outputs project data-files for display and manipulation. The "Display" process obtains graphical data to draw digital maps with proper symbols, scales, and position. "Manipulate" then takes place, with the help of the graphics and all available tools. This process can launch query, analysis, and simulation processes. The input of "Manipulate" can also come from the Function class of the Forestry application which may concern factors such as growth, cut, and plant of trees, as well as natural processes such as wind and fires. The output of "Manipulate" either updates the graphics display or is sent to "Present". The result buffer stores any results from the "Present" process which regulates presentation formats. The results can be delivered in any meaningful manner.

All processes in Figure 6.7 are high-level ones which need to be expanded. For example, the display process may be broken down into generalizing, centring, clipping, etc. which require the knowledge of window size, location, and other relevant information. The most complex one is perhaps the "Manipulate" process which may be further specified with any lower processes that help users to achieve goals. Most object classes attached to the data flow diagram come from the object model. Some of the objects are treated as data stores for the reason that only trivial operations are expected from them. Which object can be attached to which processor should be based on the class accessibility provided by the object model. The Result buffer was not in the object model. It represents a temporary object allocated by the system. Its object structure should be defined based on the possible types of results.

6.5 The Software Architecture

While the object, dynamic, and functional models concentrate on individual aspect of a system, the *software systems architecture* describes how the whole system needs to be decomposed into software components. The architecture concerns aggregating of objects with common functionality into subsystems, defining of communication topology between the subsystems, and their inter-processing interfaces. The decision about grouping classes into a component needs to involve the following factors: 1) There are classes close to users and classes close to hardware and hardware-dependent software. The system should be structured such that tasks can be fulfilled without users worrying too much about the internal complexity close to the machine level. Intermediate classes between users and machine dependent facilities should be formed if necessary. 2) The construction of the software architecture needs to observe trade-offs between flexibility and efficiency. A flexible component is hardware independent and may be used in various programming environments. It can also survive future changes or extensions to its internal structure and functionality. However, flexibility is usually achieved at the cost of performance because additional software layers have to be used to hide hardware dependent instructions and to

support modularity. This implies overheads when executing instructions from higher level layers to lower ones. 3) The decomposition of a complex system may produce largely, but not entirely, independent components. This means that some software components may not be functioning without the presence of other supportive ones.

The software industrial trend is moving toward client-server architectures with service providers wrapped as software components whose interfaces are well-designed to allow informed access with different privileges. Modification to the internal states and structures of components are restricted. A component is often a subclass of some superclass. Therefore most common services designed for the superclass can be directly inherited or customized. This encourages reuse of code and contributes to shorter development periods.

The software systems architecture for FORMONET is illustrated in Figure 6.8. It follows the principle of a *layered* systems design [Rumbaugh et al. 1991, pp. 200-201]. All software components are organized into three layers. The top layer is the user layer which is composed of a GUI interface supported by a set of virtual GIS, SDSS, and application modelling tools. Users can be responsible for customizing the appearance of the interface, using publicly available components and tools. The second layer is exposed to application developers who are domain experts and have knowledge about the functions and interfaces of the VMO-server. The components at this layer provide functionality support to the GUI interface. The services that require database and expert systems support are provided by the VMO-server. The bottom layer is the most important one, which develops the VMO-workspace engine, the management of metadata about the federated database, and the expert systems engine. Developing internal services is the responsibility of systems designers and developers. Internal services are linked to the VMO-server which further exports them to the layer immediately above it. The VMO-server provides a level of abstraction over location and implementation of the components below it. Through the VMO-server, application programmers obtain functional support from the databases using standard interfaces. How and where this support is provided become transparent.

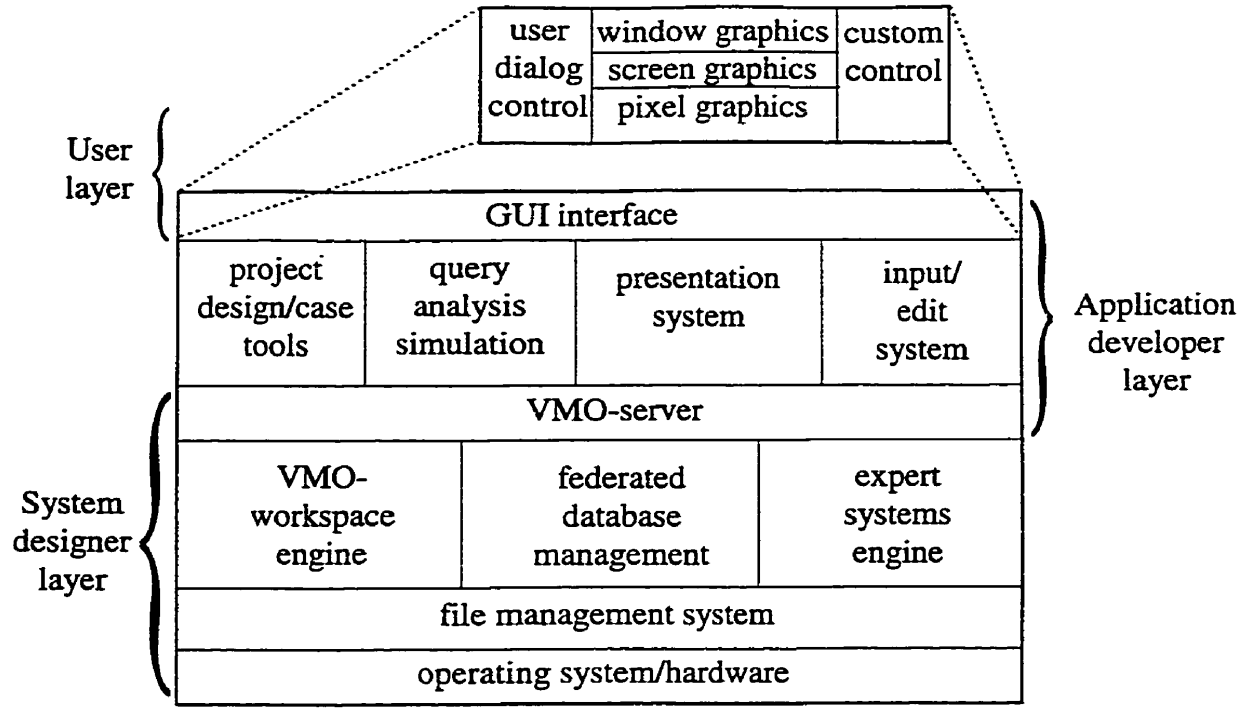


Figure 6.8 The software systems architecture for FORMONET

In summary, this chapter follows the OMT method to provide a general design of the forestry data management software system (FORMONET) based on the VMO model. The OMT spans analysis, systems design, and object design phases. The central content of the OMT methodology is emphasized on the production of object, dynamic, and functional models. The object model of the FORMONET is discussed in view of the problem statement and domain knowledge. Relevant classes are extracted through the analysis of the problem, together with the associations among them. One of the associations establishes the linkage between the WORKSPACE class and the VMO classes. In addition to diagrams showing structures of objects, descriptions and the data dictionary are all important part of the object model. The dynamic model aims to capture events stimulated externally and actions of objects in response to messages. Instead of entailing individual scenarios of events and interactions, this chapter presents a general event flow diagram for FORMONET, which encompasses major classes and accommodates most types of user stimulated events. An example of the state diagrams for the VMO server interacting an event is also given in this chapter. The functional model consists of diagrams showing data

values transforming through processes. As with the dynamic model, a general functional model for FORMONET is presented in this chapter. The functional model involves several major processors each of them need to be decomposed. Nevertheless, the functional model demonstrates the accessibility of data stores and actors modelled in the class analysis. Given software components covered in the three models, a three-layer software architecture is presented at the end of the chapter, with the expectation that the system is flexible to the addition of new components and to modifications. With this architecture, developers at each layer can concentrate on the problems more relevant to that layer.

6.6 Relationship to the Research Objectives

In closing this chapter, it is desirable to examine how the research objectives are achieved through the operation of the prototype system. In other words, we need to question if the FORMONET GIS, once implemented, will possess the advanced features specified in Chapter 1.

The first feature concerns capabilities for the system to support spatial objects, to allow cartographic operations and topological analysis over the spatial objects. This feature is ensured through the inclusion of the GEO-DATABASE and especially, the WORKSPACE classes. Since the WORKSPACE supports VMO objects, topological analysis can be performed through the Voronoi construction. Cartographic operations such as colouring, applying symbols to, and labelling objects are trivial but need to be implemented. The main restriction to cartographic capabilities is the changing of projections of VMO objects. It appears necessary to reconstruct the Voronoi diagram after objects are re-projected.

Spatial concepts, the second feature, are supported through the provision of spatial data types in Chapter 5. The VMO model distinguishes two types of areal objects: maps and regions. These areal objects can be irregular in shape. Hierarchical spatial structures are recursively entailed through the containing of other maps. With the topological

implementation, spatial searches and analysis can be done intelligently - by always knowing neighbours. In the scenario of forestry application, the map object can be used to establish management zones which allow team work. Repetitive work over a whole larger area can be avoided. At the operational level, maps can be used to combine forest stands together to form blocks based on which analyses are applied.

With the implementation of the VMO components for distribution, a forest management project can use the WORKSPACE to collect maps residing on different servers. Maps can be constructed and maintained by responsible teams working on management zones. This feature not only allows the whole project to be operated on updated data, but frees higher level managers from worrying technical details involved in individual maps so that they can concentrate on more important work. On the other hand, a VMO map may be used by multiple projects and users. The theoretical base of the VMO model and the current software technology demonstrate the feasibility of the client/server architecture for the VMO COM objects. Further design and implementation are needed to achieve the goal.

The dynamic feature of the FORMONET, through its connection to VMO servers, allows maps to be updated in short periods, and any updates can be seen by users (clients) of the data components. During the maintenance, the map server will be temporarily blocked, clients can however continue to work on other area of the container map.

As is discussed in Chapters 4 and 5, the VMO wraps geometry and topology of spatial data, and can be provided with interfaces to access GIS functions over each data type. The implementation of functions can be hidden from users. This feature ensures the reusability of the software code and compatibility of the software components versioning through time. The requirement for this feature is that the VMO component must be designed following the COM specification.

Finally, the incremental process of the VMO topological structure makes it possible to track down the map history. The tracking process can also be accompanied by topological queries

at any point of time, hence topological history of the evolving objects is also tractable. A scenario of applications of this feature finds the auto-correlation between the yield of forest product and silviculture operations on forest stands over a long time frame. As was pointed out in 3.7, the full temporality of the database using the VMO can only be realized after more structures are designed for complex objects.

With the above, we would say that most of the research objectives are attainable through the application of the VMO model to forestry. The implementation of the dynamic Voronoi construction, spatial searches and GIS operations, the forward/backward reconstruction of the Voronoi maps, and the partitioning/pasting of the Voronoi diagrams in a prototype system should support this conclusion. What left to be done, however, are the implementation of the VMO model with the Common Object Model (COM) specification and architecture, and the deployment of the VMO COM objects with the FORMONET application design.

Chapter 7

Other Applications of the VMO Model

7.1 Introduction

Most GIS projects involve large quantities of complex spatial data. With the development of data acquisition techniques, the accumulation of widely dispersed digital data resources, and the improvement of telecommunication technology, the availability of GIS data has steadily increased compared with some years ago. Accordingly, the worry about the initial cost of data collection has been reduced while the demands become more acute as to how to get data into GIS applications so as to best represent reality quickly. There are several problems which are worth noting for future GIS data modelling and development:

Rapid prototyping. By rapid prototyping is meant the ability for GIS application developers to formulate and test their initial solutions to application problems at an early stage of the software life-cycle. This requires software engineering techniques and a visual programming environment to specify object, dynamic, and functional models of the problem without first considering the detailed implementation. The purpose of prototyping is to make sure that all components are necessary and function as an integrated whole. In running a prototyped system many problems may emerge and be treated at early stages, instead of being found only after large amounts of effort have been spent. Rapid prototyping in GIS applications would be enhanced if the required spatial data types and operations are modelled and wrapped as components which are readily included into problem specifications.

Dynamizing. Dynamizing refers to the ability to update a modelled spatial database when new data become available. This capability is very useful in whatever area when dynamic changes to spatial configurations are observed and need to be modelled and analysed

promptly. Besides forestry applications, which motivated the research in this thesis, telecommunication and route navigation applications have emerged favouring this feature. The transmitting of signals for cellular telecommunication and way-finding in automobiles are affected largely by urban dynamic environment [e.g. Lee 1995 pp.103-156]. Modelling of these changes must be realized within a realistic time-frame.

Parallel processing. Computing spatial problems in parallel presents a futuristic attraction to GIS applications which handle very large quantities of data and which require real-time responses. Modelling forest fires, digital terrains, and forest watersheds belongs to this kind of applications. In order to support parallel processing, a large spatial problem has to be decomposed into subproblems, which presents a key challenge to workers of geographical information systems. Parallelization has not been considered as of priority in the major stream of GIS development today. The author feels this situation will be changed as computers with multiprocessors become affordable, as application needs become more acute, and of course, as competition of processing power in GIS industry becomes reality.

Automated map generalization. Automated map generalization is a long-standing problem in GIS development which was traditionally related only to cartographic representation of map objects. In reality, cartographic generalization alone has been very difficult. The problems in traditional map generalization include: most algorithms are applied in isolation to individual objects; localized neighbourhood relationships are not included, which often raises errors and conflicts; the results are unfavourably sensitive to the changes of scaling factors; the resulting map is fixed on a uniform scale; and data certainty information is not usually utilized and therefore the data uncertainty of a resulting map is left uncontrolled. Because of these problems, recent work on map generalization seeks integrated solutions which encompass all aspects of GIS from data modelling to the use of GIS products [Mark 1991; Müller et al. 1995; Weibel 1995; Joao 1995].

Cognitive process. An increasingly important problem with GIS development and applications, as with other computerized systems, is how much human intelligence can be artificially incorporated into a system. This may includes allowing data modelling

processes to consider mental models of space; understanding and mimicking cognitive processes in a problem solving environment; modelling and updating incomplete domain knowledge; presenting easy-to-understand results; and communicating with users using semantically meaningful dialogues [Head 1984, Hayes-Roth, 1985; Eastman 1985; Blades and Spencer 1986; Moulin 1990; Edwards 1991; Kuhn 1996].

The dynamic VMO model is potentially useful for the first two problems listed above. The main supporting argument lies in the very modularity of objects. That is, every container object is equipped with topology and geometry about its components, and dynamic methods to maintain these properties. These object modules can be wrapped naturally as classes of components to be used in an application programming environment. Including a VMO object is as simple as including a dialogue box. It is possible to allow a map object to be modified at either design or run time. During designing and running an application, detailed spatial components and their attributes can be filled. Application designers do not have to worry about implementing algorithms to manipulate these objects and to obtain values derived from spatial properties. This feature, that combines dynamics and modularity of spatial objects, makes it substantially different from traditional ways of handling spatial data.

The VMO model may also be appropriate for tackling the hard problems such as map generalization, and futuristic problems such as parallel processing and embedding "intelligence" into objects to allow cognitive treatment of spatial applications. In the rest of this chapter, we will first look into problems with parallelizing spatial processes and outline the solution using the VMO model. The integrated approach to map generalization is then presented, which includes a preliminary attempt to incorporate an intelligent map agent to oversee the map generalization and map use processes. These are not completed applications, but they suggest future directions.

7.2 Parallel Processing of Spatial Problems

Parallel processing means to compute tasks in parallel, in contrast to processing instructions in series with the von Neumann architecture. The hardware requirement must include a computer with more than one processor. The number of processors in processor arrays varies dramatically. A coarse-grained parallel computer can have between two and several hundred processors, while a fine-grained one contains anywhere between several hundred and many thousands of small processors [Mower 1992]. Multiprocessor architectures also fall into two extremes, the share-everything and share-nothing architectures [Özsu and Valduriez 1991].

In a *share-everything* architecture (Figure 7.1), any processor has access to any main memory component or disk unit through a fast interconnection. Since every single access to a data item requires access to the common interconnect, such architectures may suffer from a communication bottleneck caused by contention for the interconnection. One way to solve the problem is to have a limited number of powerful processors. In the context of database management, meta-information (e.g. the directories to data and the data dictionary) and control information (e.g. the lock table in concurrent access control) can be shared by all processors.

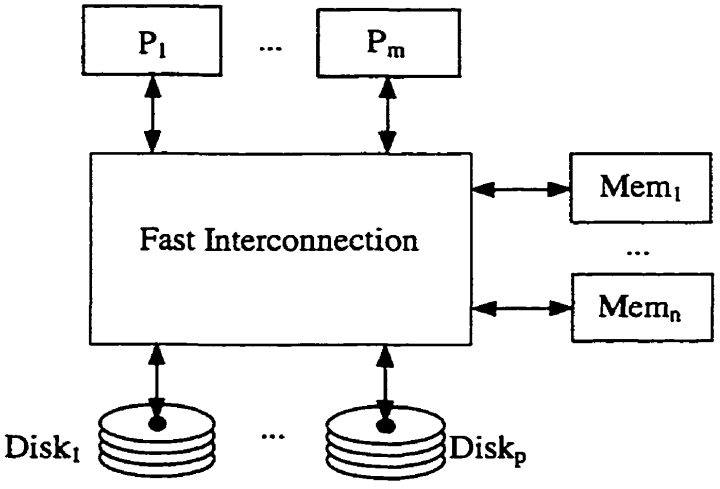


Figure 7.1 Share-everything architecture (after Özsu and Valduriez [1991])

In a *share-nothing* architecture (Figure 7.2), each processor has exclusive access to one or more memory components and one or more disk units. A node with this architecture includes a processor, a local cache memory, and a disk unit on which resides a local database. Diskless nodes may be used to interface with application servers or to process intermediate computation in parallel. The term share-nothing refers to the fact that there is no sharing of main memory or disks by the nodes. The only shared resource is the interconnection, with which nodes can exchange messages. This architecture can be viewed as a particular implementation of a distributed database system. The main idea is that a powerful computer may be built out of several smaller and less powerful ones. One similarity with the distributed database approach is that each node can be managed by the same local system. Therefore, each node must implement solutions to the global data directory, distributed data definition and control, distributed query processing, and distributed transaction management. However, the major difference with a distributed database system is that a node of the multiprocessor is not a site at which a user can run an application program. Application programs run typically on an application server and interface the multiprocessor system through a specific communication channel.

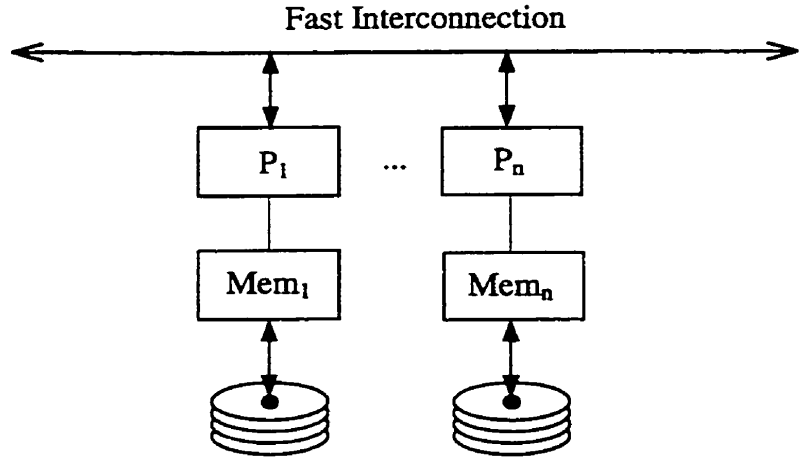


Figure 7.2 Share-nothing architecture (after Özsu and Valduriez [1991]

The share-nothing architecture is more able to achieve two important objectives: performance and extensibility. Performance improvement is obtained by using two

complementary solutions. First, data should be carefully fragmented across many nodes so that parallelism is maximised when processing a distributed query. Second, distributed data management should be efficiently supported by a distributed database operating system. The main difficulties are the partitioning of the data so that most of the queries get processed in parallel, and the development of efficient parallel algorithms for performing database operations. Extensibility is the ability to smoothly increment the growth of the system by adding new nodes. A share-nothing architecture is uniform and thus extensible. Furthermore, the same architecture and same system can be used for a large range of database sizes.

Declustering and assigning data and operations to processors are major concerns in designing algorithms for parallel processing. This includes determining the size of clusters to each processor in order to obtain optimized performance. In the context of spatial applications, it is often the case that operations run by one processor needs to get information passed from processors handling its spatial neighbours. Performance of parallel processing is largely dependent on whether neighbouring processors (in multi-dimensional processor arrays) correspond to neighbouring spatial clusters, as all messages must compete for the right-of-way through common communication channels.

In view of the structure of the VMO model, it naturally favours the share-nothing architecture. The data declustering has been taken care of by the partitioning technique. One immediately thinks of assigning each map object to one processor. The topological structure embedded in the map object can help to preserve the processor-object neighbourhood relationships. Two possible problems with this database oriented partition based on the natural spatial configuration are: 1) a processor may not be large enough to handle one map object; 2) unbalanced assignment may result, which contributes to either very busy or idle processors. More research is needed on this respect of the problem.

7.3 Automated Map Generalization

This section presents a system approach to tackle problems involved in automated map generalization. The objectives are to combine database generalization and dynamic object generalization capabilities in the system, and to couple a map agent on top of a map object which constructs user maps for navigating, performs tasks on behalf of, and communicates with, the users. The object classes are topologically and geometrically structured with the dynamic VMO-tree. This approach is based on a popular consensus that automated map generalization is actually part of a fundamental problem in GIS development, and a satisfactory solution cannot be achieved without an integrated consideration of database modelling, artificial intelligence methods, individual object generalization algorithms, and object-oriented technology [Müller et al. 1995; Weibel 1995; Buttenfield 1991, 1995; Keller 1995; Nyerges 1991; Mark 1991].

7.3.1 The Schematic View of Automated Map Generalization

Automated map generalization is a complex decision-making process which must be steered by goals and rules from the geographical application domain, so that the generalized representation conveys knowledge consistent with reality. Taking generalized map production as an integral part of a GIS, it can be used as early as during the geo-database modelling, all the way the interactive process of querying and presenting information in a problem-solving environment where a map is intelligently used (Figure 7.3).

7.3.2 Database Generalization

Database generalization utilises data modelling formalisms to capture the map structure of applications at a given point of time. The formalism describing geometric objects and their relationships has been discussed in previous chapters. The construction of the VMO structure reflects an information abstraction process in which higher level objects represent outline views of geographic spaces and details can be examined in lower level objects with successively finer resolutions (Figure 7.4).

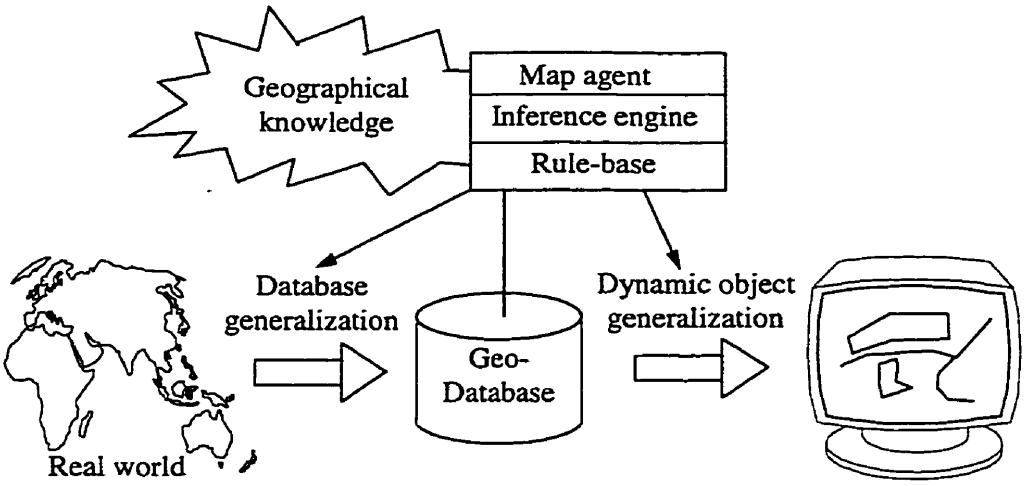


Figure 7.3 A schematic view of generalization

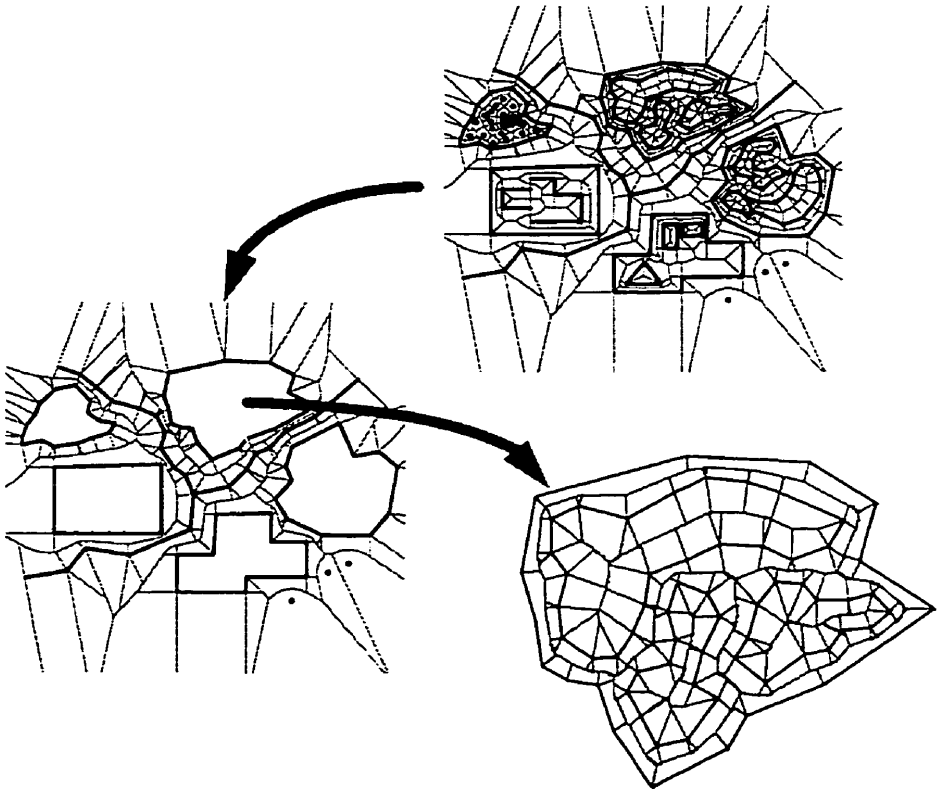


Figure 7.4 Hierarchical generalization of VMO objects

An important step to construct a generalized geographic database is to recognise geometric structures that correspond to geographical phenomena [Mark 1989]. This requires applying on-line knowledge representation techniques to catch structural and process knowledge [Nyerges 1991]. Inference rules need to be devised for generating structural knowledge [Buttenfield 1991] based on neighbourhood reasoning in a geographical context. The dynamic feature and the object-oriented design of the VMO model enhance triggering appropriate procedures and rules at the right time and place. The Voronoi diagram is proved to be a reliable local structure to reason and extract perceptual structures from atomic objects [Ahuja and Tuceryan 1989]. An experiment in utilising the Voronoi diagram for analyzing building clusters was reported recently [Regnauld 1996].

7.3.3 Map Agents

The relatively static database design incorporates geographic knowledge and cartographers' intuition on structural constraints of maps. The dynamic database generalization process would rely on both declarative and procedural knowledge and, especially, on mechanisms to integrate and enrich both types of knowledge through the evolution of the database. A map agent serves such a the mechanism in an interactive environment. An artificial agent is an object which possesses formal versions of a mental state, and in particular formal versions of beliefs, capabilities, choices, commitments, and a few other qualities [Wooldridge and Jennings 1994; Shoham 1994]. Autonomous agents are primarily developed from the field of distributed artificial intelligence (DAI) and play active roles in a dynamic environment of a complex system which involves multiple, co-operative decisions by different autonomic components with goals.

The map agent is an object class whose state is a repository of declarative metadata about the mapping classes and rules derived from geographical and cartographic expert knowledge. An example of generalization rules of a bay is given in Mark [1991]. Functions of a map agent constitute the inference engine conducting knowledge collection and representation. The inference engine contains the logic to control and direct search and reasoning techniques. The logic techniques can be characterised as having four basic parts

[Michaelsen et al. 1985]: 1) selection of the relevant rules and data elements; 2) matching the active rules against data elements to determine which rules have been triggered, indicating they have satisfied the antecedent condition; 3) scheduling which triggered rules should be fired; and 4) executing (firing) of the rule chosen during the scheduling process. The choice of an appropriate control strategy to address these four actions is dictated by the problem under consideration, the content of the object database, and the structure of the knowledge base.

There is also a technical reason for coupling a map agent on top of the geo-database. A geo-database must have classes of geometric objects as well as thematic objects. It is desirable that complex geometric objects are implemented with some generalization capabilities such as that they know how to generalize themselves before being presented graphically at a given scale. It would be inappropriate to include geographical knowledge in the generalization procedure because one geometric object may be dynamically associated with different geographical objects. On the other hand, a thematic object is a conceptual one which refers to, but is not, a geometric object. It would be awkward to include specific generalization rules in the thematic object because of loss of generality. A map agent understanding both thematic and geometric models can serve as the co-ordinator between them.

In the context of map generalization, the primary role of the map agent is to control, schedule, and validate dynamic generalization operations enacted by objects. More about dynamic object generalization is explained in the next section. Another imperative role of the map agent is to aid map uses. To achieve success in automated map generalization, the purposes of mapping must be understood and intended users must be integrated [Dymon 1989; Blades and Spencer 1987]. Typical uses of maps include navigation, measurements, and visualization (of landscape patterns) [Head 1984], for the purposes of spatial analysis, planning, designing, simulation, and decision making. We argue here that of the three uses of a map, the most important one is navigation. The other two functions rely on the result of navigating a map (albeit in a broad sense). In a computer environment where data of a map are stored in a database, navigation means to intelligently "search" through the database for

useful information. Navigation differs from the familiar search functions in that: 1) Navigation is not driven by precisely defined parameters such as the range of a search, or specific object types expected; it is rather driven by goals which are formulated from problems and which may change over time. 2) Navigation is accompanied by some memory models to store knowledge and reasoning abilities to influence decisions about what information would be useful and where to find the information. 3) Unlike search functions with simplistic objectives, the navigation process needs to be structured into a model of a few components. The functions of these components would include automatically defining objectives and performing interchangeable outline and detailed search processes. One of the main purposes of the navigation model is to allow users to concentrate on their problems, with less distraction, and to finish tasks with non-surplus and sufficient information.

Generating a schedule for a generic navigation model in an autonomous fashion would be a highly desirable goal of the map agent. For this purpose, the map agent needs to perform communication. This includes communication between computer systems, between a system and humans, and among humans with various levels of knowledge. Studies show a strong dissatisfaction concerning current maps ability to convey knowledge. The difference between the functional abilities of geographic information systems and the expectations of users is tremendous. Maps developed in current spatial databases may not be the same as those understood by a user. For a specific task, a user may have her/his own interpretation of the space and construct her/his own mental models for the task. The mental models or mental maps are centred with respect to the observer. With a goal specified, a mental map is constructed through a repetitive process, exchanging and updating information between short term and long term memory spaces. Psychological studies show that cognitive models of spaces in a human brain form hierarchical structures and that a more generalized perception about a space is obtained by moving his eyes along more detailed small parts of the space [Steinke 1987; Eastman 1985; Blades and Spencer 1986; Dobson 1985; Head 1984].

Programming a map agent in order to mimic map users presents a long-standing challenge. Although the geo-database briefly described earlier represents progressively generalized views of the world, it reflects only one state of limited expert knowledge. How to navigate through the structure and construct different views to present to various users can not be answered very easily. We hope that understanding and developing map agents will lend a hand to solve such problems.

7.3.4 Dynamic Object Generalization

Dynamic object generalization is the process activated before an object is drawn. How many and what objects to draw is the decision of the map agent in accordance with the purpose of the map use. When an object receives the message to draw itself, it invokes the built-in generalization procedure to prepare the graphics data. The container map object controls the generalization by providing search functions to detect any positional conflicts (Figure 7.5a), or topological errors (Figure 7.5b) of a proposed generalization with other existing objects to be drawn. Furthermore, if an object has a property describing its certainty, an uncertainty band (corridor) can be superimposed to control the quality of the generalization (Figure 7.5c). Searching functions and making corridors can be performed easily with the support of native geometric and topological structures (in our implementation, the Voronoi diagram). The function and data flows of the dynamic object generalization process may look like the one in Figure 7.6.

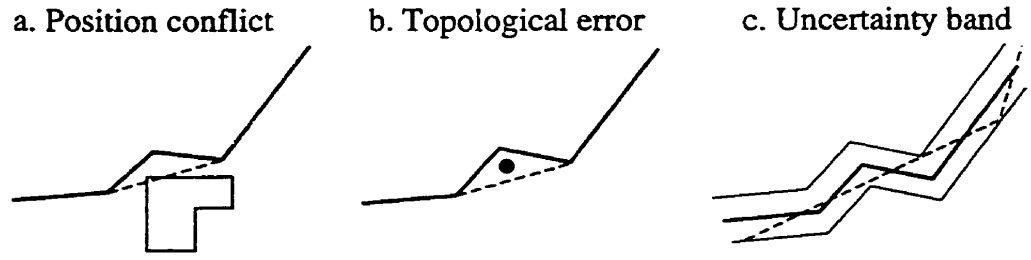


Figure 7.5 Detecting conflicts and errors, and controlling uncertainty in dynamic object generalization

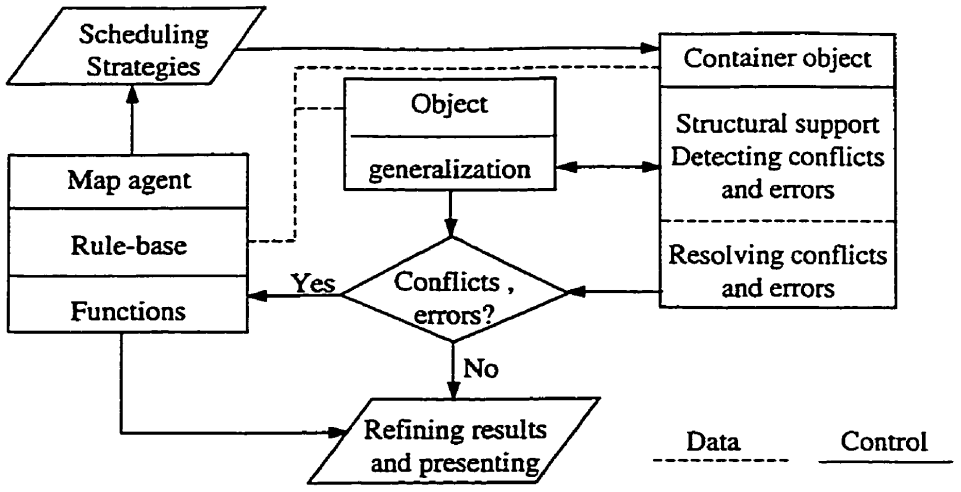


Figure 7.6 Function and data flows of dynamic object generalization

One of the advantages of having dynamic object generalization is that a map can have different scale versions simultaneously in one representation, that is, more details in the area of interest with a more outlined global picture elsewhere to visualise landscape patterns. Because generalization procedures are built in each geometric object, individual messages concerning the resolution of a generalization can be passed to the object concerned.

In summary of this section, we have discussed that although the problem of automated map generalization appears to come from the "hardware" limitation of mapping media, it has in fact a strong intellectual background and the true solution of it is never simple. The system needs to include individual generalization techniques and, more importantly, a mechanism of representing and inducing inference rules to support the choice of decisions when one individual method alone fails. The final result, i.e., a generalized map (which may even not need be drawn), must convey an understandable message to help the user to complete tasks. The system approach incorporating database generalization through off- and on-line data modelling, dynamic object generalization, and map agents looks promising for providing desirable solutions.

Chapter 8

Conclusions and Future Work

8.1 Conclusions

In the final chapter of this thesis, we take a last look at the problems stated and objectives set in the first chapter. As shown in 6.6, many of the original objectives are attainable for forestry. In addition, the research result is useful to other applications, as outlined in Chapter 7. The research and development presented leads to the following conclusions:

1. Motivated by the needs of forestry data management, the thesis sets its prime objective to designing an advanced spatial data model. The data model is aimed to be suitable for constructing a spatial database management system to support decisions concerning sustainable forestry. In order to satisfy the practical needs and achieve the objectives of the research, we started with investigating what is really meant by "sustainable forestry development" and major factors that may stand out as "characteristics" of the sustainable forestry data management system. The characteristics of sustainable forestry management strongly suggest that the data management system must be: i) of large spatio-temporal scale; ii) dynamic to changes caused by new data, goals, and technology; iii) able to support management and operations at strategic, managerial, and operational levels; iv) allow multi-user access to the data and information which are created and maintained by different owners; v) be geographically distributed, with integrated global meta-information; and vi) be flexible to data modelling, analysis, and simulation. In addition, effort was made to understand general problems and processes involved in decision-making. The philosophy and concepts discussed in this matter have helped the development of the thesis and should influence the future development of a SDSS.

2. Following the initial investigation, the basics and tools commonly used for data modelling and database development were examined (Appendix B). After this, we discussed the special properties of spatial objects, which may be represented in a variety of ways. It was found that the current prevailing practice of spatial data handling does not treat the issues well. The prevailing spatial data management systems are based on a hybrid architecture in which the geometry and topology are modelled separately. The geometric reference framework, resulted from a decomposition of the space, is rigid in that it does not generally care about the extent, shape, and complexity of the embedded objects. The topological representation of spatial objects, often based on a planar graph, does not support the geometric decomposition scheme and is not dynamic. The globalized construction of a topological structure makes the hybrid architecture difficult to apply to a distributed, federated spatial database. Besides, the architecture does not support truly hierarchical objects and object-oriented data modelling and therefore is not flexible to user needs. The analysis concludes that the hybrid data model finds itself difficult to satisfy the requirements identified for an integrated sustainable forestry data management and decision support system. We do not yet fully understand geographical spaces and as a consequence, a convincing theory which can be used to describe them is still in question.

3. Fortunately, there do exist some tools worth developing: the concept of Voronoi diagrams and the idea of dynamically constructing them. It was demonstrated that the very concept of "neighbours" centred in Voronoi diagrams catches the essential property of the constructs for geographical spaces and that the concept combines both field- and object-based views embedded in geographical modelling. Based on this, this thesis reviewed and explored important properties of the dynamic Voronoi diagrams and its dual, the Delaunay triangulation. The data structures for, and the kinematic incremental process of, constructing a Voronoi diagram were explained in length. The representation of the data model integrates the geometric object definition and preserves the fundamental neighbourhood relationship from which other topological properties of spatial objects can be conveniently derived. Using this representation, GIS operations are realized with ease, and in a topologically informed fashion. The topologically

informative operations on spatial objects form the basis of intelligent data models. Intended to use the Voronoi diagram as a dynamic GIS data model, this thesis also extended the coverage of the temporal ability required in contemporary spatial data handling. Algorithms for typical GIS operations were elaborated in the light of their graph-theoretic equivalence and the combined view of space using the Voronoi data model.

4. The thesis identified problems with the primitive Voronoi diagrams of points and line segments. These problems prevent the Voronoi data model from being applied to large data volumes and being treated in modular and hierarchical fashion in terms of complex spatial objects. A spatial object condensation technique was devised during the thesis research which led to an elegant resolution of these problems. The technique partitions a Voronoi diagram along natural container boundaries into disjoint and independent subdiagrams, in terms of spatial structures, the storage of their representations, and their functions. There is no restriction on the shape and size of containers. Smaller Voronoi diagrams constructed separately can be pasted together to define a larger Voronoi diagram. The partition uses a time proportional to the number of line segments forming the partitioning boundary to break the dependency between neighbouring subdiagrams on the memory model. The overall partitioning algorithm is affected in the worst-case by $O(n)$ in both time and storage. The spatial object condensation can also be applied to partition any triangulation networks along designated triangular edges forming closed polygons. This application produces independent memory modules of subtriangulations, such that a large triangulation network can be stored and worked with separately. The outcome of the spatial object condensation technique breaks through conventional ways of spatial data modelling and brings up a novel paradigm of construction and management of large spatial databases.

5. The extended application of the spatial object condensation technique in this thesis is the design of a dynamic spatial object database scheme called the Voronoi Map Object (VMO) model. Covered by the VMO model are formal definitions of basic geometric object classes and a description about spatial relationships between these objects. The

implementation of geometric and topological structures within a map object turns it into a VMO. With a VMO, all previously developed operations based on the dynamic Voronoi diagrams are inherited. The embedding of geometry and topology, as well as dynamic operations within a VMO, makes it stand out as an independent object class to be used in geographical applications. The VMO model is enhanced naturally by the VMO-tree which can be constructed via both top-down and bottom-up approaches, with the support of the spatial object condensation technique. The constrained VMO-tree forms a global organization about VMOs which may be geographically distributed. The thesis also speculated about operations over VMO-trees and their nodes. These operations cover issues related to distributed spatial database management systems supported by the client-server architecture.

6. In response to the needs of sustainable forestry data management and decision support systems, the thesis outlined a design of such a system based on the VMO model. The design was based on the Object Modelling Technique (OMT) which covers object, dynamic, and functional models. The object model comes from a problem statement and constitutes structural support to the other two models. The relationship between system objects and the object in charge of dynamic spatial database management using the VMO model is placed and described. In the dynamic model, the interactions between objects through dynamic events were illustrated. The data and information flow between sequentially invoked processors was demonstrated in the functional model. The software architecture of the system was devised, which organizes the system components into three layers each of which is intended for different developers. Although the whole design was at a rather coarse stage, it nevertheless sets up an integrated framework from which detailed elaboration can be completed in later development.

7. It is hoped that the VMO model can be applied in many different ways, corresponding to contemporary requirements for spatial data handling. It is especially useful in situations that require dynamic topology and integrated spatial data management systems whose database is constructed and managed by different teams at various sites.

The thesis briefed two application examples: parallel processing and automated map generalization envisaged from their database, dynamic, and intelligent aspects.

8.2 Original Contributions of This Research

The research covered in this thesis is based on the primitive dynamic Voronoi data model developed by Dr. Gold, without which the enhanced treatment of this thesis would be impossible. Theories and applications of the original dynamic Voronoi data model have been widely published in various national and international journals. The context of the early model covers the basic algorithms and implementation of the incremental and kinematic construction of Voronoi diagrams of points and line segments; the idea of the log file structure for preserving history of the construction; applications of the dynamic Voronoi diagrams in digital terrain modelling, intelligent navigation over and through spatial structures; and most of the general algorithms for GIS operations used in this thesis.

The author of the thesis has been greatly benefited by the opportunity of participating in the implementation of the primitive Voronoi data model. This experience and numerous discussions with Dr. Gold and other researchers working in this area helped to form the shape of the spatial object condensation technique and the VMO model advanced in this thesis. Specifically, the thesis brings up the following contributions to the Voronoi based GIS research and development.

1. Provided the proof for Property 4.6 concerning the configuration types of the three objects from which Voronoi vertices are calculated.
2. Complemented the description, in an algorithmic fashion, of the incremental and kinematic construction process for the Voronoi diagram of points and line segments.
3. Described and implemented the log file structure, and the forward- and backward-reconstruction of the dynamic Voronoi diagram.

4. Designed and implemented algorithms for network analysis (depth-first and breadth-first searches, minimum spanning tree, shortest path) and range search, based on the Voronoi data structure. The algorithm for polygon overlay was elaborated.
5. Designed and implemented algorithms for the spatial object condensation technique along the boundaries of containers. The method of partitioning triangular networks was described.
6. As a by-product of the spatial object condensation technique, the flaw in the current theory and practice of Voronoi diagrams of line segments in dealing with weakly-connected components is remedied.
7. The nearest-object search algorithm is modified to work for spaces with holes and weakly-connected components.
8. The VMO model is formally described, which encompasses the geometric object classes and spatial relationships, the VMO class, the data structure, the constraints, and the constructions and operations.
9. A design for a forestry data management software system is outlined, covering the object, dynamic, and functional models, and the software architecture.
10. The thesis describes the development of an operational prototype, as demonstrated by all of the Voronoi diagram illustrations used in this thesis.

8.3 Suggested Future Work

The objectives of the thesis were to define the appropriate spatial model, not to implement a full system. Therefore, the VMO model is by no means complete, either in theory, implementation, or applications. The thesis has nevertheless set up the skeleton of the data model. It is suggested that the following efforts need to be made in future work:

1. Experimenting and analyzing the condensation algorithms against large volumes of real, preferably forestry data.
2. The development of a dynamic configuration capability for automatically identifying partitioning boundaries for clusters of spatial objects. Without an automated clustering

or grouping process, the partitioning boundaries have to be manually identified or to be selected through some SQL queries.

3. Completing the VMO class design and implementation for wrapping it into software components. The interface design and implementation needs to follow the COM specification. After VMO COM objects are available, applications can be rapidly developed by plugging and playing with these components. With the current COM technology, remote access and operations on the VMO objects can be enabled without being noticed by users.
4. Detailed design and implementation of FORMONET. The production of the prototype system can set up an example for the deployment of the VMO components.
5. Researching the temporal aspects for the VMO model and implementing spatio-temporal queries over the model with different versions of VMO objects. The log file structure and forward/backward reconstruction mechanism provided in this research works on the primitive objects. The real temporality of the database using the VMO can only be realized after more structures are designed for the other VMO objects.
6. Completing the formal description for topological relationships and operations. This is felt to be important, as with other data models, for the discourse and communication with a clearly defined language.

References

Abel, D. J. and Mark, D. M. 1990. A comparative analysis of some two-dimensional orderings." *International Journal of Geographic Information Systems, 4(1),* 22-31.

Ahuja, N. and Tuceryan, M. 1989. Extraction of early perceptual structure in dot patters: integrating region, boundary, and component gestalt. *Computer Vision, Graphics, and Image Processing 48.* 304-56.

Andleigh, P. K. and Gretzinger, M. R. 1992. *Distributed Object-Oriented Data-Systems Design.* Prentice Hall.

Anthony, R.N. 1965. *Planning and Control Systems: A Framework for Analysis.* Harvard University Press, Boston.

Aurenhammer, F. 1991. Voronoi diagrams - a survey of a fundamental geometric data structures. *ACM Computing Surveys, 23(3),* 345-405.

Bédard, Y., Gagnon, P. D., and Vallière, D. 1994. *Le formalisme MODUL-R 2.01 et le dictionnaire de données pour la conception des bases de données spatio-temporelles.* Research document, Centre for Research in Geomatics, Laval University.

Bédard, Y. and van Chestein, Y. 1995. La gestion du temps dans les systèmes de gestion de données localisées: état actuel et avenues futures. *Proc. International Symposium of Geomatics V (CIG Montreal branch) – The Road to Innovation.* Nov. 9-10, Montreal, Canada, 21-33.

Beeri, C., Bernstein, P. A. and Goodman, N. 1978. A sophisticate's introduction to database normalization theory. *Proc. 4th Int. Conf. Very Large Data Bases,* 113-24.

Bentley, J. L. 1975. Multidimensional binary search trees used for associative searching." *ACM Communications, 18 (9),* 509-17.

Bertolotto, M. and de Floriani, L. 1994. Multiresolution topological maps. In Molenaar, M. and de Hoop, S. (eds.) *Advanced Geographic Data Modelling: Spatial Data Modelling and Query Languages for 2D and 3D Applications. New Series No. 40,* Delft, The Netherlands, 179-90.

Birch, K., Blair, R., Bradley, B., Bormann, B., Browning, A., Collopy, M., Franklin, K., Manning, V., and Monesmith, J. 1993. *Interim Concepts for Developing an Adaptive Management Process.* Unpublished Report.

Blades, M. and Spencer, C. 1987. How do people use maps to navigate through the world? *Cartographica, 24(3),* 64-75.

Blades, M. and Spencer, C. 1986. The implication of psychological theory and methodology for cognitive cartography. *Cartographica, 23(4),* 1-13.

Boissonnat, J.-D., Devillers, O., Schott, R., Teillaud, M., and Yvinec, M. 1992. Applications of random sampling to on-line algorithms in computational geometry. *Discrete & Computational Geometry, 8,* 51-71.

Bollobás Béla, 1979. *Graduate Texts in Mathematics 63: Graph Theory - An Introductory Course.* Springer-Verlag, New York.

Bormann, B. T., Brookes, M. H., Ford, E. D., Kiester, A. R., Oliver, C. D., and Weigand, J. F. 1994. A framework for sustainable-ecosystem management. *USDA Forest Service General Technical Report PNW-GTR-319,* Pacific Northwest Research Station, Portland, Oregon.

Boudriault, G. 1987. Topology in the TIGER file. *Auto-Carto, 8,* 258-69.

Brown, K.Q. 1979. Voronoi diagrams from convex hulls. *Information Processing Letters 9*, 223-28.

Burrough, P. A. 1986. *Principles of Geographical Information Systems for Land Resources Assessment.* Oxford: Clarendon Press.

Buttenfield, B. P. 1995. Object-oriented map generalization: modelling and cartographic considerations. In Müller, J.-C., Lagrange, J.-P., and Weibel, R. (eds.) *GIS and Generalization: Methodology and Practice*, Taylor & Francis, 91-105.

Buttenfield, B. P. 1991. A rule for describing line feature geometry. In Buttenfield, B.P. and R.B. McMaster (eds) *Map Generalization: Making Rules for Knowledge Representation.* Longman, London. 150-71.

Cameron, M.A. and Abel, D.J. 1996. A problem model for spatial decision support systems. In Kraak, M. J. and Molenaar, M. (eds) *Proc. 7th International Symposium on Spatial Data Handling*, Delft, The Netherlands: Taylor & Francis, 3A: 25-36.

Candian Forest Service 1994. *Towards a New Era in Sustainable Forestry - The Canadian Forest Service Strategic Plan for Science and Technology: 1995-2000.* Natural Resources Canada, Ottawa, Canada.

Caron, C. 1991. Nouveau Formalisme de Modélisation Conceptuelle Adaptéaux SIRS. Masters thesis, Département des Sciences Géomatiques, Université Laval.

Chen, P. P. 1976. The entity-relationship model toward a unified view of data." *ACM Transactions on Database Systems, 1(1),* 9-35.

Chew, L. P. 1989. Constrained Delaunay Triangulation. *Algorithmica, 4*, 97-108.

Chew, L. P. 1987. Constrained Delaunay Triangulation. *Proc. 3rd ACM Symposium on Computational Geometry,* 215-22.

Chrisman, N. R. 1975. Topological information systems for geographic representation. *Auto-Carto 2*, 246-51.

Chrisman, N. R. 1978. Concepts of space as a guide to cartographic data structures. In Dutton, G. (ed.) *Proc. 1st International Advanced Study Symposium on Topological Data Structures for Geographic Information Systems*, Cambridge, MA: Harvard Laboratory for Computer Graphics and Spatial Analysis. 1-19.

Christenson, C. O. and Voxman, W. L. 1977. Aspects of Topology, Monographs and Textbooks in Pure and Applied Mathematics. Marcel Dekker, Inc. New York.

Codd, E. F. 1970. A relational model of data for large shared data banks. *ACM Communication*, 13, 377-87.

Comerford, N. B., Cole, D. W., Dyck, W. J. 1994. Impacts of harvesting on long-term site quality: future research. In Dyck, W. J., Cole, D.W., and Comerford, N. B. (eds.) *Impacts of Forest Harvesting on Long-Term Site Productivity.* London: Chapman & Hall. 363-68.

Couclelis, H. 1992. Beyond the raster-vector debate in GIS. In Frank, A. U., Campari, I., and Formentini, U. (eds.) *Theories of Saptio-Temporal Reasoning in Geographic Space, Lecture Notes in Computer Science 639*, Springer-Verlag, 65-77.

Cowen, D. J. 1988. GIS versus CAD versus DBMS: what are the differences? *Photogrammtric Engineering and Remote Sensing, 54(11),* 1551-5.

Davies, C. and Medyckyj-Scott, D. 1996. GIS users observed. *International Journal of Geographic Information Systems, 10(4),* 363-84.

de Floriani, L., Marzano, P., and Puppo, E. 1993. Spatial queries and data models. In Frank, A. U. and Campari, I. (eds.) *Spatial Information Theory - A Theoretical Basis for GIS,*

COSIT'93, Marciana Marina, Elba Island, Italy. *Lecture Notes in Computer Science 716*, Springer-Verlag, 113-38.

Devillers, O., Meiser, S., and Teillaud, M. 1992. Fully dynamic Delaunay triangulation in logarithmic time per operation. *Computational Geometry Theory and Applications, 2*, 55-80.

Dobson, M. W. 1985. The future of perceptual cartography. *Cartographica, 22(2)*, 27-43.

Doucet, B. 1990. *Développement d'un Prototype d'un Systeme d'Information à Référence Spatiale pour la Gestion des Données d'Aménagement Forestier à la Forêt Montmorency*. Masters thesis, Département des Sciences Géomatiques, Université Laval.

Drysdale, R. L. 1979. *Generalized Voronoi Diagrams and Geometric Searching*. PhD Thesis, Department of Computer Science, Stanford University.

Drysdale, R. L. and Lee, D. T. 1978. Generalized Voronoi diagram in the plane. *Proc. 16th Annual Allerton Conference on Communications, Control and Computing*, 833-42.

D'Silva, E. and Appanah, S. 1993. Forestry management for sustainable development. *EDI Policy Seminar Report No. 32*. The World Bank, Washington, D. C.

Dyck, W. J. and Cole, D. W. 1994. Strategies for determining consequences of harvesting and associated practices on long-term productivity. In Dyck, W. J., Cole, D.W., and Comerford, N. B. (eds.) *Impacts of Forest Harvesting on Long-Term Site Productivity*. London: Chapman & Hall. 13-40.

Dymon, U. 1989. Do we really know our map users? *Cartographica, 26(3&4)*, 49-58.

Eastman, J. R. 1985. Graphic organization and memory structures for map learning. *Cartographica, 22(1)*, 1-20.

Edelsbrunner, H. 1986. Edge-skeletons in arrangements with applications. *Algorithmica 1*, 93-109.

Edwards, G. 1991. Spatial knowledge for image understanding. In Mark, D. M. and Frank, A. U. (eds.) *Cognitive and Linguistic Aspects of Geographic Space*, Kluwer Academic Publishers, 295-308.

Egenhofer, M., Clementini, E., and di Felice, P. 1994. Topological relations between regions with holes. *International Journal of Geographic Information Systems, 8(2)*, 129-42.

Egenhofer, M. and Franzosa, R. 1991. Point-set topological spatial relations. *International Journal of Geographic Information Systems, 5(2)*, 161-74.

Faloutsos, C., Sillis, T., and Roussopoulos, N. 1987. Analysis of object oriented spatial access methods. *ACM SIGMOD, 16(3)*, 426-39.

Felten, V. 1998. *Integration de la Dimension Temporelle dans les SIRS, Application a la Forêt Montmorency*. Technical report, Université Laval, also Mémoire présenté en vue de l'obtention du diplôme d'ingénieur E.S.G.T. (Ecole Superieure des Geometres et Topographes), France.

Fortune, S. 1987. A sweepline algorithm for Voronoi diagrams. *Algorithmica, 2*, 153-74.

Fortune, S. 1986. A sweepline algorithm for Voronoi diagrams. *Proc. 2nd ACM Symposium on Computational Geometry*, 313-22.

Frank, A. 1983. Storage methods for space related data: the field-tree. *Tech. Rep. Bericht No. 71*, Eidgenössische Technische Hochschule Zürich.

Frank, A. and Barrera, R. 1989. The field-tree: a data structure for geographic information system. In Günther, O. and Smith, T. (eds.), *Design and Implementation of Large*

Spatial Databases, Proc. SSD'89, Santa Barbara, California, Lecture Notes in Computer Science 409, Berlin: Spriger-Verlag, 29-44.

Franklin, J. F. 1989. Toward a new forestry. *American Forestry, 95*, 37-44.

Fry, J. P. and Sibley, E. H. 1976. The Evolution of Database Management Systems. *ACM Computing Survey, 8*, 7-42.

Gabriel, K. R. and Sokal, R. R. 1969. A new statistical approach to geographic variation analysis. *Systematic Zoology, 18*, 259-78.

Gagnon, P. 1993. Concepts fondamentaux de la gestion du temps dans les SGDL. Masters thesis, Département des sciences géomatiques, Université Laval.

Giblin, P. J. 1977. Graphs, Surfaces and Homology - An Introduction to Algebraic Topology. London: Chapman and Hall.

Gold, C. M. 1997. Simple topology generation from scanned maps. *Proc. ACSM/ASPRS - Auto-Carto 13*, vol. 5, Seattle, Washington, April, 7-10, 337-46.

Gold, C. M. 1994. The interactive map. In Molenaar, M. and de Hoop, S. (eds.) Advanced Geographic Data Modelling: Spatial Data Modelling and Query Languages for 2D and 3D Applications. Netherlands Geodetic Commission Publications in Geodesy, New Series, 40, 121-28.

Gold, C. M. 1992a. An object-based dynamic spatial model, and its application in the development of a user-friendly digitizing system. *Proc. 5th SDH'92*, Charleston, U.S.A., 495-504.

Gold, C. M. 1992b. The meaning of "neighbours". In *Theories and Methods of Spatio-Temporal Reasoning in Geographic Space, Lecture Notes in Computer Science 639*, Springer-Verlag, Berlin, 220-35.

Gold, C. M. 1991. Problems with handling spatial Data - the Voronoi approach. *CISM Journal 45(1)*, 65-80.

Gold, C. M. 1990. Spatial data structures: the extension from one to two dimensions. In Pau, L.F. (ed.) *Mapping and Spatial Modelling for Navigation*, NATO ASI Series 65, Berlin: Springer-Verlag, 11-39.

Gold, C. M. 1989. Surface Interpolation, Spatial Adjacency and GIS. In Raper, J. (ed.), *Chapter 3: Three Dimensional Applications in GIS*. Francis&Taylor, London, 21-35.

Gold, C. M. 1977. The practical generation and use of geographic triangular element data structures. *Advanced Study Symposium on Topological Data Structures for Geographic Information Systems*, Cambridge, Mass.: Laboratory for Computer Graphics and Spatial Analysis.

Gold, C. M. 1976. Triangular element data structures. The University of Alberta Computing Services Users Applications Symposium Proceedings, Edmonton, Canada. 43-54.

Gold, C. M., Charters, T. D., and Ramsden, J. 1977. Automated contour mapping using triangular element data structures and an interpolant over each triangular domain. *Computer Graphics, 11*, 170-75.

Gold, C. M. and Condal, A. R. 1994. A spatial data structure integrating GIS and simulation in a marine environment. *Marine Geodesy, 18*, 213-28.

Gold, C. M., Nantel, J., and Yang, W. 1996. Outside-in: an alternative approach to forest map digitizing. *Int. J. Geographical Information Systems, 10(3)* 291-310.

Gold, C. M., Remmele, P. R., and Roos, T. 1997. Fully dynamic and kinematic Voronoi diagrams in GIS. *Algorithmica*, in press.

Gold, C. M., Remmele, P. R., and Roos, T. 1995. Voronoi diagrams of line segments made easy. *Proc. 7th Canadian Conference on Computational Geometry*, Quebéc, Canada, 223-28.

Gold, C. M. and Roos, T. 1994. Surface modelling with guaranteed consistency - an object-based approach. In Nievergelt, J., Roos, T., Schek, H.-J., and Widmayer, P. (eds.) *IGIS'94: Geographic Information Systems, Proc. International Workshop on Advanced Research in Geographic Information Systems*, Monte Verità, Ascona, Switzerland, *Lecture Notes in Computer Science 884*, Springer-Verlag, 70-87.

Goodchild, M. F. and Grandfield, A. W. 1983. Optimizing raster storage: an examination of four alternatives. *Proc. Auto Cartography: International Perspectives on Achievements and Challenges*, Ottawa, Canada, 400-07.

Gorry, G. A. and Morton, S. 1975. A framework for management information systems. In Rappaport, A. (ed.) *Information for Decision Making: Quantitative and Behavioral Dimensions* (2nd edition). Prentice-Hall Englewood Cliffs, New Jersey, 16-30.

Green, P. J. and Sibson, R. 1977. Computing dirichlet tesselations in the plane. *Computer Journal*, 21, 168-73.

Guibas, L., Mitchell, J. S. B., and Roos, T. 1991. Voronoi diagrams of moving points in the plane. *Proc. 17th Intl. Workshop on Graph, The Theoretic Concepts in Computer Science*, Fischbachau, Germany, *Lecture Notes in Computer Science 570*, Springer-Verlag, 113-25.

Guibas, L. and Stolfi, J. 1985. Primitives for the manipulation of general subdivisions and the computational of voronoi diagrams. *ACM Transaction on Graphics, 4(2)*, 74-123.

Günther, O. 1988. Efficient structures for geometric data management. *Lecture Notes in Computer Science, No. 337*, Berlin: Springer-Verlag.

Günther, O. and Wong, E. 1989. The arc tree: an approximation scheme to represent arbitrary curved shapes. In Litwin, W. and Schek, H.-J. (eds.), *Foundations of Data Organization and Algorithms*. Paris, France, *Lecture Notes in Computer Science 367*, Springer-Verlag, 354-70.

Güting. R. H. 1988. Geo-relational algebra: a model and query language for geometric database systems. In Goos, G. and Hartmanis, J. (eds.) *Advances in Database Technology - EDBT'88, Lecture Notes in Computer Science 303*, Springer-Verlag, 506-27.

Güting, R. H. and Schilling, W. 1987. A practical divide-and-conquer algorithm for the rectangle intersection problem. *Information Sciences, 42*, 95-112.

Guttman, A. 1984. R-trees: a dynamic index structure for spatial searching." *ACM SIGMOD, 13*, 47-57.

Hayes-Roth, F. 1985. Rule-based systems. *Communications of the ACM, 28(9)*.

Head, C. G. 1984. The map as natural language: a paradigm for understanding. *Cartographica, 21(1)*, 1-32.

Herring, J. R. 1991. The mathematical modelling of spatial and non-spatial information in Geographical Information Systems. In Mark, D. M. and Frank, A. U. (eds.) *Cognitive and Linguistic Aspects of Geographic Space*, Kluwer Academic Publishers, 313-50.

ITTO (International Tropical Timber Organization) 1992. *ITTO Criteria for the Measurement of Sustainable Tropical Forest Management.* ITTO Policy Development Series 3. Yokohama.

Jagadish, H. V. 1990. Linear clustering of objects with multiple attributes. *ACM SIGMOD 19(2)*, 332-42.

Joao, E. M. 1995. The importance of quantifying the effects of generalization. In Müller, J.-C., Lagrange, J.-P., and Weibel, R. (eds.) *GIS and Generalization: Methodology and Practice*, Taylor & Francis, 183-93.

Keller, S. F. 1995. Potentials and limitations of artificial intelligence techniques applied to generalization. In Müller, J.-C., Lagrange, J.-P., and Weibel, R. (eds.) *GIS and Generalization: Methodology and Practice*, Taylor & Francis, 135-47.

Kent, W. 1991. A rigourous model of object reference, identity, and existence. *Journal of Object-Oriented Programming*.

Kirkpatrick, D. G. 1979. Efficient computation of continuous skeletons. *Proc. 20th Annual IEEE Symposium on Foundations of Computer Science*, 18-27.

Knuth, D. E. 1973. *The Art of Computer Programming III: Sorting and Searching*. Addison Wesley, Reading, MA.

Kruskal, J. R. Jr. 1956. On the shortest spanning subtree of a graph and the traveling salesman problem. *Problem. Am. Math. Soc., 7(1),* 48-50.

Kuhn, W. 1996. Handling data spatially: Spatializing user interfaces. In Kraak, M.-J. and Molenaar, M. (eds.) *Proc. 7th International Symposium on Spatial Data Handling*, Delft, The Netherlands. Taylor & Francis, 877-93.

Lamont, R. H. 1988. Avoiding surprises in selecting and setting up a geographic information system. In Heit, M. and Shortreid, A. (eds.) *GIS Applications in Natural Resources*. GIS World, Inc. 1991. 71-4.

Langran, G. 1992. *Time in Geographic Information Systems*. Taylor & Francis, New York.

Lee, D. T. and Drysdale, R. L III. 1981. Generalized Voronoi diagrams in the plane. *SIAM Journal of Computing, 10,* 73-87.

Lee, D. T. and Schachter, B. J. 1980. Two algorithms for constructing the Delaunay triangulation. *International Journal of Computer and Information Sciences, 9(3),* 219-42.

Lee, William C. Y. 1995. *Mobile Cellular Telecommunications: Analysis and Digital Systems*. 2nd edition, McGraw-Hill Inc.

Mark, D.M. 1991. Object modelling and phenomenon-based generalization. In Buttenfield, B. P. and McMaster, R. B. (eds.) *Map Generalization: Making Rules for Knowledge Representation*. Longman, London. 103-18.

Mark, D. M. and Goodchild, M. F. 1986. On the ordering of two-dimensional space: introduction and relation to tesseral principles. In Diaz, B. and Bell, S. B. M. (eds.) *Proc. the Tesseral Workshop No. 2*, Swindon: Natural Environment Research Council, 179-92.

Mason, R. O., Jr. 1975. Basic concepts for designing management information systems. In Rappaport, A. (ed.) *Information for decision making: quantitative and behavioral dimensions (2nd edition).* Prentice-Hall Englewood Cliffs, New Jersey, 2-16.

Maser C. 1994. *Sustainable Forestry: Philosophy, Science, and Economics*. St. Lucie Press, Delray, Beach, FL.

Matsuyama, T., Hao, L. V., and Nagao, M. 1984. A file organization for geographic information systems based on spatial proximity. *Computer Vision, Graphics and Image Processing, 26,* 303-18.

Mehlhorn, K. 1984. *Data Structures and Algorithms 3: Multi-Dimensional Searching and Computational Geometry*. Springer-Verlag.

Michaelsen, R.H, Michie, D., and Boulanger, A. 1985. The technology of expert systems. *BYTE Magazine* 10. 303-12.

Mladenoff, D. J. and Pastor, J. 1993. Sustainable forest ecosystems in the northern hardwood and conifer forest region: Concept and management. In Aplet, G. H., Johnson, N., Olson, J. T., and Sample, V. A. (eds.) *Defining Sustainable Forestry*. Island Press, Washington, D.C. 145-80.

Molenaar, M. 1989. Single valued vector maps - A concept in geographic information systems. *Geo-Informationssysteme, (2)1*, 18-26.

Morton, G. M. 1966. A Computer Oriented Geodetic Data Base, and a New Technique in File Structuring. Unpublished report, IBM Canada Ltd.

Morris, L. A. and Miller, R. E. 1994. Evidence for long-term productivity change as provided by field trials. In Dyck, W. J., Cole, D. W., and Comerford, N. B. (eds.) *Impacts of Forest Harvesting on Long-Term Site Productivity*. London: Chapman & Hall. 41-80.

Moulin, B. 1990. *Knowledge Representation and Conceptual Modelling*. Revised version, Department d'informatique, Université Laval.

Mower, J. E. 1992. Building a GIS for parallel computing environments. In Corwin, E. and Cowen, D. (eds.) *Proc. 5th International Symposium on Spatial Data Handling*, Columbus, OH: International Geographical Union, 219-29.

Muller, D. E. and Preparata, F. P. 1978. Finding the intersection of two convex polyhedra. *Theoretical Compter Science, 7(2)*, 217-36.

Müller, J. C., Weibel, R., Lagrange, J.-P., and Selge, F. 1995. Generalization: state of the art and issues. In Müller, J.-C., Lagrange, J.-P., and Weibel, R. (eds.) *GIS and Generalization: Methodology and Practice*, Taylor & Francis, 3-17.

National Round Table on the Environment and the Economy 1993. *Forest Round Table on Sustainable Development, a Pregress Report*. Ottawa, Canada.

NCGIA 1989. The research plan of the National Centre for Geographic Information and Analysis. *International Journal of Geographic Information Systems, 3*, 117-36.

Nievergelt, J., Hinterberger, H., and Sevcik, K. C. 1984. The grid file: an adaptable, symmetric, multikey file structure. *ACM Transactions on Database Systems, 9(1)*, 38-71.

Nyerges, T. L. 1991. Representing geographical meaning. In Buttenfield, B.P. and R.B. McMaster (eds) *Map Generalization: Making Rules for Knowledge Representation*. Longman, London. 55-85.

Okabe, A., Boots, B., and Sugihara, K. 1992. *Spatial Tessellations: Concepts and Applications of Voronoi Diagrams*. John Wiley & Sons.

Orenstein, J. A. 1983. *Data Structures and Algorithms for the Implementation of a Relational Database System*. PhD Thesis, McGill University.

Ottmann, T. and Wood, D. 1986. Space-economical plane-sweep algorithms. *Computer Vision, Graphics, and Image Processing, 34*, 35-51.

Özsu, M. T. and Valduriez, P. 1991. *Principles of Distributed Database Systems*. Prentice-Hall, Englewood Cliffs, New Jersey.

Peucker, T. K. and Chrisman, N. 1975. Cartographic data structures. *American Cartographer, 2(1)*, 55-69.

Peuquet, D. J. 1984. A conceptual framework and comparison of spatial models. *Cartographica, 21(4)*, 66-113.

Preparata, F. P. and Shamos, M. I. 1985. *Computational Geometry: An Introduction*. Springer-Verlag.

Prim, R. C. 1957. Shortest connection networks and some generalizations. *The Bell System Technical Journal, 36*, 1389-401.

Proe, M. F., Rauscher, H. M., and Yarie, J. 1994. Computer simulation models and expert systems for predicting productivity decline. In Dyck, W. J., Cole, D. W., and Comerford, N. B. (eds.) *Impacts of Forest Harvesting on Long-Term Site Productivity*. London: Chapman & Hall. 151-86.

Proulx, M.-J. 1995. Développement d'un Nouveau Langage d'Interrogation de Bases de Données Spatio-Temporelles. Masters thesis, Département des sciences géomatiques, Université Laval.

Puppo, E. and Dettori, G. 1995. Towards a formal model for multiresolution spatial maps. In Egenhofer, M. J. and Herring, J. R. (eds.) *Advances in Spatial Databases, Proc. SSD'95, Lecture Notes in Compter Science 951*, Springer-Verlag, 152-69.

Regnauld, N. 1996. Recognition of building clusters for generalization. In Kraak, M.J. and M. Molenaar (eds) *Proc. 7th International Symposium on Spatial Data Handling*, Delft, The Netherlands: Taylor & Francis, 4B.1-14.

Roos, T. 1991. *Dynamic Voronoi Diagrams*. Ph.D Thesis, Universität Würzburg, Switzerland.

Rosenberg, J. B. 1985. Geographical data structures compared: a study of data structures supporting region queries. *IEEE Transaction. on Computer Aided Design, 4(1)*, 53-67.

Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., and Lorensen, W. 1991. *Object-Oriented Modelling and Design*. Prentice Hall, Englewood Cliffs, New Jersey.

Samet, H. 1990. *The Design and Analysis of Spatial Data Structures*. Addison-Wesley, Reading, Mass.

Samet, H. 1984. The quadtree and related hierarchical data structures. *Computing Surveys, 16(2)*, 187-260.

Sample, V. A. Realizing the potential of remote sensing and GIS in ecosystem management planning, analysis, and policymaking. In Sample, V. A. (ed.) *Remote Sensing and GIS in Ecosystem Management*. Island Press, 346-52.

Schmitt, I. and Saake, G. 1995. Managing object identity in federated database systems. In M. P. Papazoglou (ed.) *OOER'95: Object-Oriented and Entity-Relationship Modelling. 14th Int. Conference*, Gold Coast, Australia. *Lecture Notes in Computer Science 1121*, Springer, 400-11.

Sedgewick, R. 1992. *Algorithms in C++*. Addison-Wesley Publishing Company, Inc.

Shoham, Y. 1994. Multi-agent research in the knobotics group. In Castelfranchi, C. and E. Werner (eds.) *Artificial Social Systems, 4th European Workshop on Modelling Autonomous Agents in a Multi-Agent World*, S. Martino al Cimino, Italy. *Lecture Notes in Artificial Intelligence 830*. Springer-Verlag. 271-278.

Sibson, R. 1977. Locally equiangular triangulations. *The Computer. Journal, 21*, 243-5.

Simon, H. A. 1977. *The New Science of Management Decision*. Prentice-Hall, Englewood Cliffs, New Jersey.

Smith, J. M. and Smith, D. C. P. 1977. Database abstractions: aggregation and generalization. *ACM Transactions on Database Systems, 2(2)*, 105-33.

Stefanakis, E. 1994. *A New Approach to Range Searching Based on the Point Representation of Spatial Objects*. Masters thesis, Department of Geodesy and Geomatics, University of New Brunswick.

Steinke, T. R. 1987. Eye movement studies in cartography and related fields. *Cartographica, 24(2)*, 40-73.

Stell, J. G. and Worboys, M. F. 1994. Towards a representation for saptial objects in diverse geometries. In Pissinou, N. and Makki, K. (eds.) *Proc. 2nd ACM Workshop on Advances in Geographic Information Systems*. National Institute for Standards and Technology, Gaithersburg, Maryland, New York: ACM Press, 28-33.

Stolfi, J. 1987. Oriented projective geometry. *3rd ACM Symp. on Computational Geometry*, 76-85.

Stuth, J. W. and Smith, M. S. 1993. Decision support for grazing lands: an overview. In Stuth, J. W. and Lyons, B. G. (eds), *Decision Support Systems for the Management of Grazing Lands: Emerging Issues*. Man and the Biosphre Series, Vol. II, UNESCO and The Parthenon Publishing Group. 1-35.

Szarmes, M. C. 1997. *Modelling the evolution of spatio-temporal databases structures for GIS applications*. Masters thesis, Department of Geomatics Engineering, The University of Calgary.

Szymansky, T. G. and van Wyk, C. J. 1983. Space efficient algorithms for VLSI artwork analysis. *Proc. 20th IEEE Design Automation Conference*, 734-39.

Toussaint, G. T. 1980. The relative neighbourhood graph of a finite planar set. *Pattern Recognition, 12*, 261-8.

Tsichritzis, D. C and Lochovsky, F. H. 1982. *Data Models*. Prentice-Hall, Englewood Cliffs, New Jersey.

US Burean of the Census 1970. *The DIME Geocoding System. Tech. Rep. 4*, Burean of the Census, Washington, DC.

van Oosterom, P. 1993. *Reactive Data Structures for Geographic Information Systems*. Oxford University Press.

Weibel, R. 1995. Three essential building blocks for automated generalization. In Müller, J.-C., Lagrange, J.-P., and Weibel, R. (eds.) *GIS and Generalization: Methodology and Practice*, Taylor & Francis, 56-69.

White, M. 1983. N-trees: large ordered indexes for multidimensional space. Presented at the Lincoln Institute of Land Policy's Colloquium on Spatial Mathematical Algorithms for Microcomputer Land Data Systems.

Wooldridge, M. and Jennings, N. R. 1994. Agent theories, architectures, and languages: a survey. In Wooldridge, M. and Jennings, N. R (eds.) *Proc. ECAI-94 Workshop on Agent Theories, Architectures, and Languages*, Amsterdam, The Netherlands. *Lecture Notes in Computer Science 890*, Springer-Verlag, 1-39.

Worboys, M. F. 1995. *GIS: A Computing Perspective*. Taylor & Francis.

Worboys, M. F. 1994. Object-oriented approaches to geo-reference information. *International Journal of Geographic Information Systems, (8)4*, 385-99.

Worboys, M. F 1992a. A generic model for planar geographic objects. *International Journal of Geographic Information Systems, (6)5*, 353-72.

Worboys, M. F 1992b. A model for spatio-temporal information. In Corwin, E. and Cowen, D. (eds.) *Proc. 5th International Symposium on Spatial Data Handling*, Columbus, OH: International Geographical Union, 602-11.

World Bank 1993. *Forestry Management for Sustainable Development*. An EDI Policy Seminar Report No. 32. The World Bank, Washington, D.C.

World Commision on Environment and Development 1987. *Our Common Future*. Oxford University Press, Oxford, UK.

Yang, W. 1992. *A New Range Searching Algorithm for Large Point Databases*. Masters thesis, Department of Geodesy and Geomatics, University of New Brunswick.

Yang, W. and Gold, C. M. 1996. Managing spatial objects with the VMO-tree. In Kraak, M.-J. and Molenaar, M. (eds.) *Proc. 7th International Symposium on Spatial Data Handling*, Delft, The Netherlands. Taylor & Francis, 711-26.

Yang, W. and Gold, C. M. 1995. Dynamic spatial object condensation based on the Voronoi diagram. *Proc. 4th Int. Sym. of LIESMARS'95 - Towards three dimensional, temporal and dynamic spatial data modelling and analysis*, Wuhan, China, 134-45.

Zonneveld, I. S. 1990. Scope and concepts of landscape ecology as an emerging science. In Zonneveld, I. S. and Formann, R. T. T. (eds.) *Changing landscapes: An Ecological Perspective*. New York: Springer-Verlag. 3-20.

Appendix A

Geographic Information and Decision Support Systems

A.1 Geographic Information Systems

Definition. A *geographic information system* (*GIS*) is usually defined as a computer-based information system that enables capture, modelling, manipulation, retrieval, analysis, and presentation of geographically referenced data [Worboys 1995, pp. 1].

The above and similar [NCGIA 1989] definitions actually summarize the common functionalities of a GIS. Other practical definitions may further emphasize typical applications of GIS. For example, Cowen [1988] argues that "A GIS is best defined as a decision support system involving the integration of spatially referenced data in a problem solving environment." There is also a tendency to include users (e.g. organizational context as described by Burrough [1986]) of a GIS in the definition. The key argument is that a GIS is a complex system the proficient use of which needs human-computer interactions. A team of trained staff with GIS skill and application domain expertise is indispensable to successful operations of GIS applications. This point of view expresses a popular concern that current GIS is not easy to understand and use. Mastery of it requires comprehensive training to know the tools it provides, the data models and data structures on which it is built, and more importantly, the weakness and limitations it brings to bear on certain applications.

GIS hardware. The general hardware components of a GIS include the computer and its peripherals (Figure A.1). The computer system must have a central processing unit (CPU), and is configured with a random access memory (RAM) as well as an internal hard disk. The peripheral devices consist of: a keyboard, a screen display monitor, a floppy disk drive, a compact disk (CD) drive, a tape drive, a digitizer, a plotter, and optionally, a scanner. The

CPU is in charge of executing instructions designated by computer programs. During the execution of an application, data and programs have to be read and temporarily stored in the RAM. One use of the hard disk is to persistently save data and program files. Floppy disks and CDs are also storage media which can be used for communication. Nevertheless, more and more communications between users of computers nowadays are performed through a network of computers. An effective network connection to internet and intranet becomes a dispensable peripheral when configuring a computer hardware for a GIS. The monitor provides the visual display of data, programs, and control panels for human-computer interaction, normally with the help of a keyboard and a mouse. Digitizers and scanners are used for converting data on maps and documents into digital form. The results of the data processing can be presented via a hard copy obtained by means of a plotter.

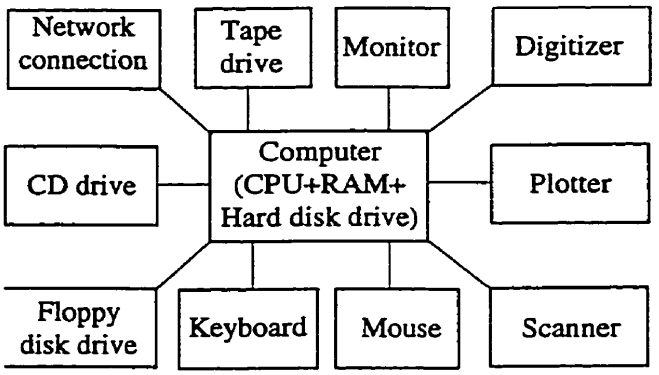


Figure A.1 The general hardware components of a GIS

GIS software. The GIS software constitutes a more important and usually more expensive component of the system, in comparison with the hardware. The basic software component is divided into closely related functional modules or sub-systems which handle data input/edit, presentation, query/analysis, data modelling and database management, and human-computer interface, respectively. A schematic view of these basic modules and the building architecture is shown in Figure A.2.

Data input/edit. A GIS is useless without data. Data can be spatial or non-spatial. Spatial data are usually supposed to represent something that can be located in the real world. Non-

spatial data in a GIS are usually subordinate. They are associated to and describe spatial data. All data, spatial or non-spatial, must be in digital forms of some formats before they are entered into a GIS. A GIS normally captures data through scanners, digitizers, and standard input peripherals as for all information systems (keyboard, mouse, CD and disk readers, internet/intranet). Each of these devices is supported by appropriate software called drivers. Commonly involved processes in this module include format conversion, error detection and correction, rectification and registration, and line simplification. Most of these processes are highly human/machine interactive. Editing data after its initial input to resolve topological errors is tedious manual work with most commercial GIS.

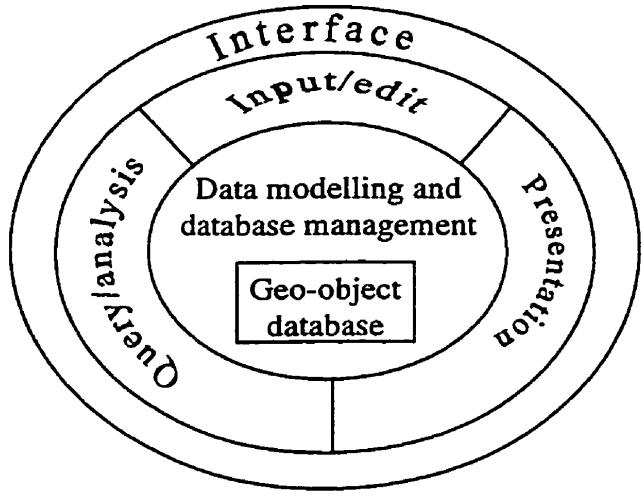


Figure A.2 The main software components of a GIS

Capturing spatial data in a GIS contributes a major cost which can be as high as 70 per cent of total GIS system expenses [Dickinson and Calkins 1988]. The digitizing process is slow and expensive, but it does deliver a well structured and information rich vector data source. Scanning is fast and cheaper, but the raster end-product is limited in structure and in the ability to add values by attaching attributes to spatial references. Two possibilities for improving spatial data capturing exist: 1) increasing automation and reliability in raster-vector conversion; 2) improving a raster-based system to provide equal or enhanced functionality with vector systems. Recent research and development in the Industrial Research Chair in geomatics applied to forestry, at Laval University has demonstrated a

great potential for reducing cost on data capturing. Three techniques have been developed: 1) interactive digitizing with dynamic topology [Gold 1994]; 2) rapid digitizing for environmental data [Gold et al. 1996]; and 3) map scanning plus automated raster-vector conversion with simple vector topology [Gold 1997].

Query and analysis. Data query and analysis constitute a major software block of a GIS. Querying a GIS for information can be thought of as the process of asking (by users) and answering (by a GIS) questions. Application related questions are numerous, notably covering three aspects of geographical data: spatial, aspatial, and temporal. Some of these general questions are the following [Burrough 1986, pp. 9]:

(a) Where is object A?
(b) Where is A in relation to place B?
(c) How many occurrences of type A are there within distance D of B?
(d) What is the value of function Z at position X?
(e) How large is B (area, perimeter, count of inclusions)?
(f) What is the result of intersecting various kinds of spatial data?
(g) What is the path of least cost, resistance, or distance along the ground from X to Y?
(h) What is at points X1, X2, ... ?
(i) What objects are next to objects having certain combinations of attributes?
(j) Reclassify objects having certain combinations of attributes.
(k) Using the digital database as a model of the real world, simulate the effect of process P over time T for a given scenario S.

Some of these and other questions can be answered easily, some with difficulty, and some cannot be answered given current theory and technology. Most of the questions inquiring about temporal properties and topological dynamics of geographical phenomena cannot be easily answered using the majority of commercial GIS on the market. Difficult questions often rely on the underlining data models and the analytical processes of a system. Analysis of geographical data usually refers to processes that need to manipulate the spatial or temporal aspects of data in order to calculate or derive the geometric, topological and

temporal properties of the data. Geometric properties of spatial data concern their shape, size, or location in an embedding space. Topological properties are the ones that are invariant under "rubber-sheet" transformations, which continuously deform the underlying space without breaking it. Examples of topological properties include: neighbourhood, adjacency, connectivity, and inclusion. Temporal properties concern the evolution, disturbance events, and precedence relationships of geographical data in a certain time frame. A part of forestland F transformed into an industrial park IP in 1990 may be an example indicating an evolution of F, and at the same time, a creation of a new entity called industrial park (IP).

Query and analysis will inevitably retrieve the geographical database for the desired data, which involves search and access operations. Retrieval, search, and access of data are often used ambiguously to mean "getting data from databases". They do, however, have distinctions. *Retrieving* is a collective term which treats a database as a blackbox without really referring to internal data organization and search operations provided by the database management. *Searching* data from databases concerns more of the logical structures of data organization and of the operations to locate relative addresses of desired data in the whole structure. The whole structure may be divided into several logically related files which reside in storage media. *Accessing* data is the set of methods which actually "touches" the physical records of the desired data, based on their relative addresses in each file. To access a data record, the file containing the data must first be opened. Other important accessing operations include find, read, delete, modify, insert. Finally, the file opened must be closed. How efficiently data in the database can be searched and accessed in order to be retrieved, and sometimes modified depends on the construction of data models and supporting data structures. Construction of data models and data structures for building a spatial database management system is the central concern of this thesis.

Data presentation. This module concerns how data and results of query/analysis from a GIS can be presented to users through output devices. Traditional general purpose databases provide output in the form of text and numbers in tabular form. A report generator is a standard feature of a DBMS and allows the embedding of database output in

a high-level document format which can be enhanced by colourful charts and graphics. GIS requires a further step to the presentation of maps and images. Such presentation should enable the user to visualize their base maps and results of analysis, which may be multi-dimensional. Information will be required at varying scales, and appropriate detail should be presented at each scale. This is closely related to the issue of *geographical database and cartographic generalization* which will be discussed in this thesis in a rather integrated way.

Geo-object database, data modelling, and database management. The geo-object database stores geographical data. The depositing of geographical data in the database, however, must follow pre-defined database structures which themselves are translations of data modelling processes with compatible database languages. The essence of data modelling is to identify and describe entities and relationships in reality, with respect to the real world processes affecting them. In the computer world, entities and relationships are often called objects and relations, respectively. Geographical data in the database need to be represented as objects with various complexities. Each geographical object (geo-object) has to be described with a few references or characteristics. Typical aspects of a geo-object may be spatial, textual/numerical, graphical, and temporal characteristics.

The spatial aspect of a geo-object concerns the specification and structure of the object in an embedding space. Geometric and topological data models are often used to study spatial structures of and operations upon geo-objects. Geometric and topological data models are often referred to collectively as spatial data models. A good spatial data model should be closed under operations on the objects concerned, and expressive with respect to a wide range of object types and relations. The value of a geo-object is primarily reflected from its textual/numerical aspect. The identification of a geo-object in a database requires a unique value together with its spatial description. The textual/numerical aspect of a geo-object is also referred to as the thematic aspect and is often studied with thematic data models which concern the semantics of objects in the application domain. The graphical aspect of a geo-object concerns how the object is presented or visualized. It was traditionally thought of merely as a cartographic issue. More and more, the approach now is to consider it an

integral part of the spatial data modelling. The temporal aspect of a geo-object concerns its changes over space and time. A meaningful change in an object occurs when its spatial or textual/numerical status is altered at a particular time. Data models must encompass a long time frame and capture meaningful changes such that the previous status of an object can be retrieved and restored if desired. Another issue associated with the temporal aspect of a geo-object is how data models can reflect changes in a localized and dynamic fashion. A data model is dynamic if integration among its components can be maintained upon changing events without restructuring the whole model.

Computerized database capabilities are enabled by a collection of programs called a database management system (DBMS). The DBMS is hence a software system that facilitates the process of defining, constructing, and manipulating databases for various applications. *Defining* a database involves specifying the types of data to be stored in the database, along with a detailed description for each type of data. *Constructing* the database is the process of storing the data itself in some storage medium that is controlled by the DBMS. *Manipulating* a database includes functions such as querying the database to retrieve specific data, updating the database to reflect changes in the real world, and generating reports from the data. A database in a GIS must be spatial, which distinguishes a GIS from other information systems. Nevertheless, like any other standard database, a DBMS in a GIS should possess the following basic capabilities:

1. Controlling redundancy: Redundancy refers to storing the same data multiple times. This happens in traditional file processing systems where each user group independently keeps data files for their own use. Redundancy leads to problems such as duplication of effort, wasted storage space, and more seriously, inducing data inconsistency. The database system controls redundancy by integrating the views of different user groups during database design which ensures that each logical data item is to be stored in one place only.

2. Sharing of data: A database may be accessed by multiple users at the same time. This is essential if data for multiple applications is to be integrated and maintained in a

single database. Sharing of data is an important way of reducing the high cost of data capturing. It also promotes inter-discipline cooperation to achieve more optimized results. By sharing data, authorized updating of data can be incorporated by multiple applications in a timely fashion. By allowing multiple access, a database system must include concurrency control methods to ensure that several users trying to update the same data do so in a controlled manner so that the result of the update is correct. Besides, security and authorization procedures must be placed to assign privileges to different users for the retrieval or updating of data.

3. Providing backup and recovery: If the computer system fails in the middle of a complex update program, the recovery subsystem is responsible for making sure that the database is restored to its state before the program started executing. Alternatively, the recovery subsystem could ensure that the program is resumed from the point at which it was interrupted so that its full effort is recorded in the database. This requires that the backup subsystem should keep track of database update operations and should record updates and data necessary for the future recovery. This capability is essential to maintaining the temporality of databases.

Some researchers feel that current geographical database systems lack proper database management support. This shortcoming leads to lack of portability, maintainability, reusability and sometimes correctness [Stuth and Smith 1993]. Over the past twenty years, most of the theoretic research on DBMS in GIS have emphasized developing appropriate spatial data models. In current practice, the majority of spatial DBMS are composed of carefully devised programs to handle the management of spatial data. These systems have gained needed functionalities by developing customized, individual stand-alone programs. Standard capabilities of a DBMS, especially the ones dealing with multiple access and update of spatial data and management of time in the database system, have not been fully realized. A satisfactory solution to these DBMS issues still relies on the promise of a modern spatial data model and open system architecture.

GIS interfaces. A GIS is not just a store for depositing geographical data. One of its purposes is to support and to facilitate the use of data. A GIS should provide a variety of user interfaces. The types of interface include query languages for casual users, programming language interfaces for application programmers, forms for parametric users, menu-driven or iconic interfaces for naive users, and natural language interfaces. On-line, context-sensitive help facilities also constitute an important part of the interface.

A.2 Spatial Decision Support Systems

Definition. A Decision Support System (DSS) can be viewed as an integrated solution to help people to make their decisions. It can encompass methodologies as diverse as computer models, expert systems, information systems, discussion groups, and structured thought and evaluation processes. The use of a DSS can improve the objectivity of decision making, especially where complex interactions are involved. A Spatial Decision Support System (SDSS) is one that involves the integration of spatially referenced data in a problem-solving environment.

One of the key characteristics of a SDSS, as with a general DSS, is that problems to be solved are often ill-defined or ill-structured. As is put by Simon [1977], "Decisions are non-programmed to the extent that they are novel, unstructured, and consequential. There is no cut-and-dried method of handling the problem because it hasn't arisen before, or because its precise nature and structure are elusive or complex, or because it is so important that it deserves a custom-tailored treatment." Seeking solutions for unstructured problems requires the integration of our understanding of the problems themselves. In these processes, additional data, information and knowledge might be required and there is no obvious stopping rule for the investigation. Understanding problems involves such aspects as identifying observable and controlling process variables, discovering casual relationships, and developing a preference structure to rank outcomes [Cameron and Abel 1996]. As the investigation proceeds, some unstructured problems may become semi-structured or structured.

In the construction of a DSS, it is important to understand the human decision making or problem-solving processes. Research on human problem solving supports Simon's claim that all cycles of problem solving can be broken down into three phases of activities: intelligence, design, and choice [Simon 1977]. The *intelligence* activity searches the environment for conditions calling for decision; the *design* process invents, develops, and analyzes possible courses of action; and the *choice* activity selects a course of action from those available. Normally, intelligence precedes design, and design proceeds choice. The cycle of phases is, however, far more complex than the sequence suggests. Each phase in making a particular decision is itself a complex decision-making process. The design phase, for example, may call for new intelligence activities; problems at any given level generate subproblems that in turn have their intelligence, design, and choice phases, and so on. These problem-solving stages all contribute to the answering of the questions: "What is the problem", "What are the alternatives", and "Which alternative is best".

Another indispensable consideration in building a DSS concerns the nature and type of decisions within an application domain. It is realized that the nature and type of decisions are closely related to the organizational structure of the application which designs, implements, and uses the DSS. For each level of the structure, decision activities and functional requirement of the system are significantly different from that at other levels. A classification scheme differentiating planning, controlling, and operating activities within an organization is developed by Anthony [1965]. This scheme consists of three categories of managerial decisions and suggests that these categories represent activities sufficiently different in kind to require the development of different systems.

The first category is *strategic planning*. It is the process of deciding on the objectives of the organization, on the changes in these objectives, on the resources used to attain these objectives, and on the policies that are to govern the acquisition, use, and disposition of these resources. This process focuses on the choice of objectives for the organization and on the activities and means required to achieve these objectives. As a result, a major

problem in this area is predicting the future of the organization and its environment. Strategic planning does not follow a routine procedure.

The second category is *management control* in which managers assure that resources are obtained and used effectively and efficiently in the accomplishment of the organization's objectives. Key aspects of activities in this area include 1) inter-personal interaction; 2) the activity takes place within the context of the policies and objectives developed in the strategic planning process; 3) the paramount goal of the activity is the assurance of effective and efficient performance.

The third category is *operational control*. It is the process of assuring that specific tasks are carried out effectively and efficiently. The boundaries between these categories are often not clear. The classification, however, helps to identify the fundamental character of information needed by different decision-making and management levels [Gorry and Morton 1975].

Strategic planning often needs aggregate information which is obtained mainly from sources external to the organization itself. Both the scope and variety of the information are quite large, but the requirements for accuracy are not particularly stringent. The nonroutine nature of the strategic planning process means that the demand for information occurs infrequently. The information needed for operational control stand in sharp contrast to those of strategic planning. The task orientation requires information of a well-defined and narrow scope. The information is quite detailed and arises largely from sources within the organization. Very frequent use is made of this information, and it must therefore be accurate. The information requirements for management control fall between the extremes for operational control and strategic planning. Much of the information in this area comes from the process of interpersonal interaction.

In addition to understanding the decision process in an organization, knowledge about information flow and assumptions made in the design of a DSS help to justify its appropriate use and hence the decision outcomes. Any information system assumes views

of the world which reflect in some way the opinions of the designer about the information needed at each stage of decision within an organization. These assumptions are intrinsic to the internal architecture of a DSS and are vital to its flexibility and success. The following diagram (Figure A.3) (modified from Mason [1975]) illustrates the information flow and general assumptions incorporated at each stage of a decision.

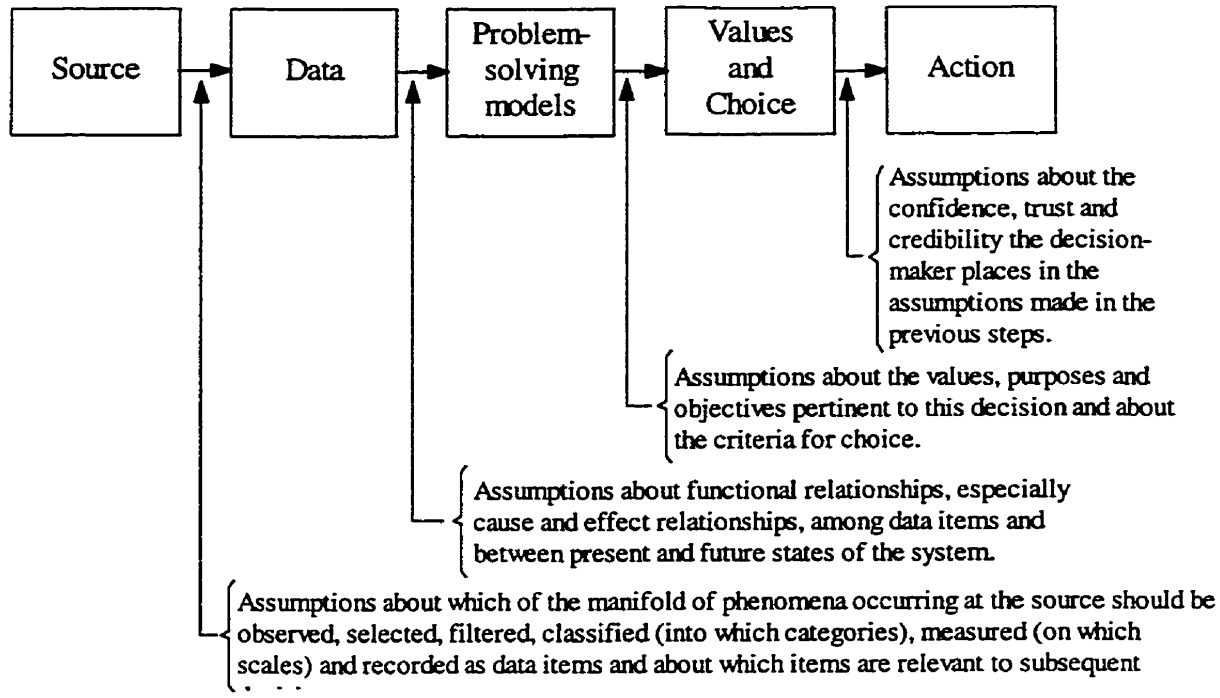


Figure A.3 Assumptions at each stage of information flow

The information flow in Figure A.3 is compatible with the three phases of the problem-solving process discussed earlier. The processes involved in identifying and collecting information sources and assimilating data correspond to the intelligence phase; the problem-solving models, which may include predicting, analyzing, simulating, and logic deducting models and tools to the design phase; and the outcome, values and choice of a decision to the choice phase.

Structured much like a GIS, a modern SDSS might in addition incorporate an expert system which runs over a rule-base for an application domain (Figure A.4). The expert system formalizes, stores, manages, and manipulates domain specific knowledge needed for

problems. The subsystem interface takes care of interactions between the geo-object database and the expert system. The integrating of the problem-solving models and tools transfers the total system to a more specific application system. Consequently, the user interface may need to be customized to suit domain experts or decision-makers who are novices to computerized systems.

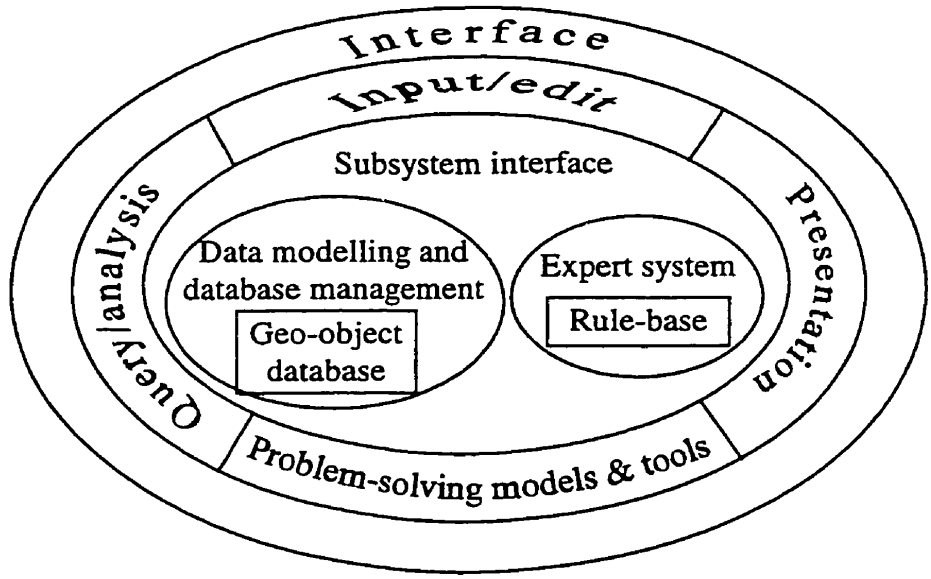


Figure A.4 The software architecture of a SDSS

Stuth and Smith [1993] reviewed the development of a SDSS for ecosystems, from an domain expert point of view. The following aspects have arisen as a wish list of a future SDSS:

Graphical interface, multimedia and data visualization. Appropriate and attractive forms of data display are a vital part of helping people to make better decisions. Good maps, graphs, flowcharts, and other displays aid people to interpret data and to appreciate its significance much more rapidly, and if honest, more objectively.

Spatial landscape analysis. Greater emphasis is being placed on seamless, application-specific spatial landscape analysis to address planning applications which have both

temporal and spatial responses. This requires spatial analyzers to be flexible enough to be linked to planning models without disrupting a planning session.

Integrating data from RS, digital photography and GPS systems. Directly importing these data sources into a SDSS can help monitor the effectiveness of management strategies imposed on landscapes.

Expert systems. Considerable interest is focusing on using specialized, well-focused expert systems embedded in SDSS to help users with parameterizing simulation models and to match technological development options with management systems.

Embedded simulation models. The core of many SDSS will always be good simulation models, varying from simple spreadsheet-style aids to complex representation of system function where data and understanding exist. Developers will need to pay greater attention to controlling model complexity for the sake of users and their input data needs while maintaining the quality and integrity of results.

Natural resource information networks. There is a growing need to create on-line data for regional, national, and international databases to service these data requirements including soils, land use, weather, plant attributes, endangered species, etc.

Open systems architectures. SDSS should meet current users needs yet have sufficient flexibility to adapt to future technology. Especially, on-line communications through the internet need to break software, hardware, and technology barriers to allow different operating systems, databases, and applications, interact with each other.

Standardization and co-operation. There is an increasing need to standardize data and applications to facilitate data exchange. For SDSS to have widespread value and ready accessibility to users, some display standards are needed.

Appendix B

Tools And Concepts In Data Modelling

The appendix discusses the role and nature of models in information modelling, from which the general phases of developing data models and methodologies applied to each stage can be presented. The development starts from the real world, through different levels of abstractions, to the implementation with a computer. The real world concerned is prescribed by the geographical phenomena occurring in a forest ecosystem with respect to the environment wherein a forest SDSS works. Special treatments for spatial information are considered within the context. The purpose of discussing basic tools and data modelling concepts in this part is to have an overall understanding of the fundamentals involved in constructing database models, instead of taking them for granted. More complex modelling structures and processes, like the ones addressed in Chapter 2, have these basic structures and simple processes.

B.1 The Role and Nature of Data and Process Models

A *model* is a human-observed representation or reflection of something in the real world. The primary reason for using a model is because the real world is often so large and complex that the comprehension of it is very difficult and sometimes impossible by a human being without the help of a model. The understanding of positional relationships between continents, for example, might be outside human capacity if some kind of overall perception (verbal description, drawing, sketch) is not present in his/her mind. This knowledge is obtained by reading a world map which is a representation with simplified contents, reduced size, and rounded accuracy. This illustrates the important use of a model for communicating knowledge and provides a basis for the study of the real world at some resolution level. A *data model* is a model concerning data. The emphasis put on a data

model is the data, i.e., the measurement, observation, and description captured from the real world. The types and characteristics of data determine how complex a modelling process must be to effectively represent the real world. The existence of data, together with a host of models, transfers the study of the real world into the study of data models of the real world.

It must be noted that studying a representation of the real world is not the sole objective of data modelling. A more interesting consideration in data modelling is that data models provide a background on which such processes as analysis, reasoning, planning and simulation can occur. In order to have a meaningful and useful data model of the real world, modelling processes occurring in the real world must proceed in parallel. This requires that the internal structures of a data model match the specific needs of process modelling.

Any representation of the real world is an abstraction. It is impossible to accommodate all aspects of the real world in the construction of a model. Important factors need to be selected. This includes identifying application-significant entities (things, concepts, events), and the relationships, processes, and constraints or conditions that delimit these components. These constrained components constitute the information content of a data model. The abstract nature of data models determines that they are not exact replicas of the real world but partial views of it. Therefore, the development of data models must contend with the inverse process: transferring results from data and process models back to the part of the real world modelled.

Questions concerning the study of a data model are: 1) which entities are covered by the model; 2) which relationships between entities are explicitly expressed; 3) what are the operations on those entities and relationships; 4) what are the constraints that bound operations and relationships, the dynamic interactions of one operation with another, as well as the interactions with entities; 5) is the data model closed under valid operations; and 6) how accurate and precise are the information contents of the model in comparison to their counterparts in the real world.

B.2 Mathematical Basics

This section introduces some basic mathematical concepts and tools useful in constructing complex models of the real world. The power of these mathematical basics lies in their abstract ability. The abstract constructions, operations, axioms and theorems can then be applied to interpreting real world situations. The most important and axiomatic concept in the construction of mathematical models comes from set theoretic notions.

B.2.1 Sets

A *set* is a collection of things (called its members or elements), the collection being regarded as a single object. Whenever possible, we will use italic capital letters as names for the sets we introduce and italic small letters for elements that may or may not be members of a particular set. If S is a set, we indicate the fact that a is an element or member of S by the notation of $a \in S$. Similarly, we write $a \notin S$ to say that a is not a member of S.

"Set" and "member" are two primitive notions within the axiomatic method of the set theory. Other concepts will be defined in terms of these two primitives which themselves will remain undefined. Instead, a list of axioms concerning the primitive notions is adopted:

Equality: If two sets, A and B, have exactly the same members, then they are equal:

$$\forall A\ \forall B[\forall x(x \in A \Leftrightarrow x \in B) \Rightarrow A = B].$$

Empty set: The set B having no members, denoted $\varnothing$, is called the empty set:

$$\exists B\ (\forall x \mid x \notin B).$$

Subset: For two sets, A and B, if every member of A is a member of B, then A is a subset of B, denoted $A \subseteq B$:

$$\forall A\ \forall B[\forall x(x \in A \Rightarrow x \in B) \Rightarrow A \subseteq B].$$

A is said to be contained in B if $A \subseteq B$. If $A \subseteq B$ but $A \neq B$, then we say that A is a *proper* subset of B and write $A \subset B$.

Power set: For any set A, there is a set B whose members are exactly the subsets of A. The set B is said to be the power set of A, denoted $\wp(A)$:

$$\forall A\ \exists B\ \forall x(x \in B \Leftrightarrow x \subseteq A).$$

B.2.2 Operations on Sets

We are going to discuss several ways of composing new sets from old ones. These operations constitute part of the algebra of sets.

Union: For two sets, A and B, their union, denoted $A \cup B$, is the set whose elements are members of either A or B:

$$A \cup B = \{x \mid x \in A \text{ or } x \in B\}.$$

Intersection: For two sets, A and B, their intersection, denoted $A \cup B$, is the set whose elements are members of both A and B:

$$A \cap B = \{x \mid x \in A \text{ and } x \in B\}.$$

Difference: For two sets, A and B, their difference, denoted A/B, is the set whose elements are members of the first set A but not the second set B:

$$A/B = \{x \mid x \in A \text{ and } x \notin B\}.$$

Relative complement: This operation is usually applied to a single argument $A \subseteq S$, where the complement of A, denoted A', is a set whose elements are members (of S) not in A:

$$A' = \{x \mid x \in S \text{ and } x \notin A\}.$$

The diagrammatic representations of the previous operations are shown in Figure B.1. The shaded regions represent the new sets resulting from the respective operations.

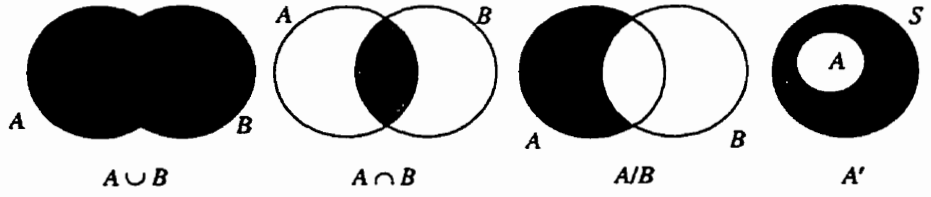


Figure B.1 Set union, intersection, difference, and relative complement

The union and intersection operations can be generalized to n arguments without regard to the order of the composition. In complete analogy with the use of the summation symbol in arithmetic, we may therefore define

$$\bigcup_{i=1}^{n} A_i = A_1 \cup A_2 \cup \cdots \cup A_n$$

$$= \{a \mid a \in A_i \text{ for some } i = 1, 2, \cdots, n\}$$

$$\bigcap_{i=1}^{n} A_i = A_1 \cap A_2 \cap \cdots \cap A_n$$

$$= \{a \mid a \in A_i \text{ for all } i = 1, 2, \cdots, n\}$$

Using the generalized union and intersection, the intuitive notion of the partitioning of a set can be defined. Before defining partition, we need to agree to say that the sets A and B are

disjoint if $A \cap B = \varnothing$ (intuitively, they do not overlap). More generally, any finite number of sets $A_1, A_2, \cdots, A_n$ is said to be disjoint (or, for the sake of clarity, mutually disjoint) if

$$A_i \cap A_j = \varnothing \ (i \neq j)$$

Partition: A collection $\pi = \{A_1, A_2, \cdots, A_n\}$ of nonempty subset $A_i \subseteq S$ is a partition of S if

(a) $\bigcup_{i=1}^{n} A_i = S$ (the A_i are a *covering* of S), and

(b) $A_i \cap A_j = \varnothing$ for all $i \neq j$ (they are mutually disjoint, no overlap)

The subsets A_i are called the *blocks* of the partition. A partition of S into k nonempty blocks is called a *k-partition* of S. Figure B.2 intuitively illustrates the partition operation.

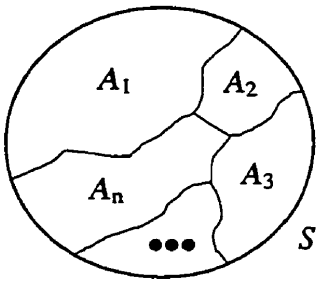


Figure B.2 A partition of the set S

B.2.3 Relations and Functions

The simplest mathematical structure of all is the unstructured set. However, sets on their own are limited in their applications in modelling. As everything in the real world is related to something else, and sets can be used to hold things, it is of paramount importance to define relational structures between sets. This can be treated with the aid of the set product construction, which in turn will lead quite naturally to the important definition of relationships and functions. Without the latter, a meaningful analysis of our subsequent mathematical structures would be impossible.

Product: Given any sets A and B, the product, denoted $A \times B$, is defined to be the collection of all ordered pairs (a, b) such that $a \in A$ and $b \in B$. That is

$$A \times B = \{(a, b) \mid a \in A, b \in B\}$$

By an *ordered* pair, we mean that (a, b) and (a', b') are regarded as equal only when $a = a'$ and $b = b'$. Thus $(a, b) \neq (a', b')$, in general. In fact, when $A = B = R$, the set of real numbers, then $A \times B$ is recognized as the set of points in the Cartesian plane. For this reason, the product $A \times B$ is quite often referred to as the *Cartesian product* of the sets A and B.

As in the case of unions and intersections, we can extend the definition of products to arbitrarily n (for $n \geq 2$) factors:

$$\begin{aligned} \prod_{i=1}^{n} A_i &= A_1 \times A_2 \times \cdots \times A_n \\ &= \{(a_1, a_2, \cdots, a_n) \mid a_i \in A_i \text{ for each } i = 1, 2, \cdots, n\} \end{aligned}$$

In the special case where all the A_i are the same, say $A_1 = A_2 = \cdots = A_n = A$, we denote this product set by A^n. Thus for any set A we have

$$A^n = \{(a_1, a_2, \cdots, a_n): a_i \in A_i \text{ for all } i = 1, 2, \cdots, n\}$$

and the elements of such a product set are usually called the *n-tuples* from the set A. The term n-tuple is just the generalized terminology: pair, triple, quadruple, $\cdots$, n-tuple.

Binary relation: A *binary relation* R from a set A to a set B is simply a subset $R \subseteq A \times B$. We say that $a \in A$ is *related* to $b \in B$ (by the relation R) if $(a, b) \in R$. The notation $a \, R \, b$ is used in the connection.

Given a relation R on a set A, the following three important properties are often used to classify relations on a set, we use the symbols x, y, z to denote members of A:

(i) *Reflexive*: $x\,R\,x$, for all x

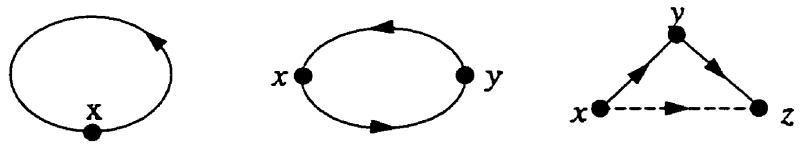
(ii) *Symmetric*: $x\,R\,y \Rightarrow y\,R\,x$

(iii) *Transitive*: $x\,R\,y, y\,R\,z \Rightarrow x\,R\,z$

(i′) *irreflexive*: $x\,R\,x$, for no x

(ii′) *antisymmetric*: $x\,R\,y, y\,R\,x \Rightarrow x = y$

The graphical representation of the three properties is given in Figure B.3, where the arrowhead edges depict the relation.



(i) Reflexive (ii) Symmetric (iii) Transitive

Figure B.3 Reflexive, symmetric, and transitive relations

The binary relation R is reflexive if it satisfies (i); symmetric for (ii); and transitive for (iii). If a relation R satisfies all the three properties, the relation is said to be an *equivalence relation*, denoted ~. The properties of an equivalence relation express important aspects of being the same, which are ordinarily taken for granted, and are usually obvious for specific equivalence relations. Equivalence relations are the primary tools employed in the process of abstraction, or selectively ignoring differences which are irrelevant to the purpose at hand. Within a given context, we say that two things are equivalent if the differences between them do not matter. Another way of looking at equivalence relations is as ways of dividing things into classes. The result of this process is a collection of new sets. For any $a \in A$, an *equivalence class* $[a]$ determined by a can be defined, which is the subset of A consisting of all the elements that are related (equivalent) to a. That is,

$$[a] = \{x \mid x \sim a\}$$

It is interesting to note that the partition of a set A induces an equivalence relation on A which relates elements of A in the same block. All elements in one block constitute an equivalence class.

Partial ordering relation (denoted by $\leq$): The relation $\leq$ is a binary relation which is transitive, reflexive, and antisymmetric. For a set A, it is called a *partially ordered set* or *poset* if the partial ordering relation is on A. Two elements a and b in A are said to be *comparable* under $\leq$ if either $a \leq b$ or $b \leq a$; otherwise they are *incomparable*. If every pair of elements of A are comparable, then we say that $[A; \leq]$ is *totally ordered* or that A is a *totally ordered set* or a *chain*. In this case, the relation $\leq$ is called a *total order*.

Function: A *function* is a special type of relation between elements of one set S and those of another T, denoted f: $S \rightarrow T$. The first set S is called the *domain*, and the second set T the *range*. The distinction of a function from a relation is that it transforms each $x \in S$ into one and only one element $f(x)$, called the *image* of x, in the range T. Thus functions are by nature single valued in that

$$f(x) \neq f(y) \Rightarrow x \neq y$$

The image is a subset of the range. The relationship between function $f(x)$, domain, range, and image is illustrated in Figure B.4.

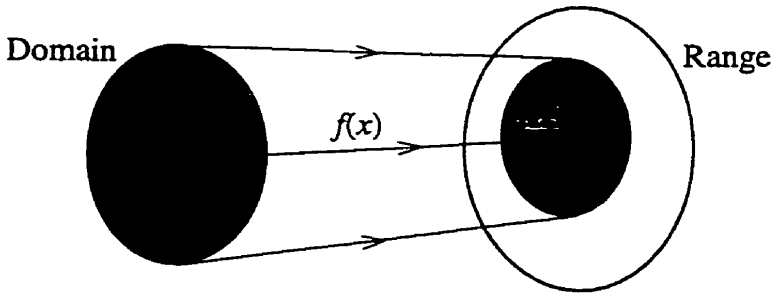


Figure B.4 Sets for $f(x)$ (adapted from Worboys [1995], pp. 109)

A function f is said to be *injective* (or *one-to-one*) if it transforms two distinct elements in the domain into two distinct elements in the range, that is

$$f(x) = f(y) \Rightarrow x = y.$$

A function f is *surjective* (or *onto*) if each $t \in T$ can be written as $t = f(s)$ for some $s \in S$, that is, if every element of the range is an image. That is

$$t \in T \Rightarrow \exists s \in S \ni t = f(s)$$

A function that is both injective and surjective is *bijective* (or *one-to-one correspondence*).

B.2.4 Graphs and Trees

Abstract graphs: A graph G is an ordered pair of disjoint sets (V, E) such that E is a subset of the set of unordered pairs of V. The set of V is the set of vertices and E is the set of edges. E can be any relation on V. $V = V(G)$ is the vertex set of G and $E = E(G)$ is the edge set. $G' = (V', E')$ is a *subgraph* of G if $V' \subset V$ and $E' \subset E$, we write $G' \subset G$. If G' contains all edges of G that join two vertices in V' then G' is said to be the subgraph *induced* or *spanned* by V' and is denoted by $G[V']$.

The *order* of G is the number of vertices; denoted by $|G|$. The same notation is used for the number of elements (cardinality) of a set: $|X|$ denotes the number of elements of the set X. Thus $|G| = |V(G)|$. The *size* of G is the number of edges; denoted by $e(G)$. We write G^n for an arbitrary graph of order n. Similarly, $G(n, m)$ denotes an arbitrary graph of order n and size m.

Two graphs are *isomorphic* if there is a correspondence between their vertex sets that preserves adjacency. Isomorphic means "having the same form". Thus $G = (V, E)$ is isomorphic to $G' = (V', E')$ if there is a bijection $\varnothing$: $V \rightarrow V'$ such that $xy \in E$ iff $\varnothing(x)\ \varnothing(y) \in E'$. Clearly, isomorphic graphs have the same order and size. Usually we do not

distinguish between isomorphic graphs, unless we consider graphs with a distinguished or labelled set of vertices (e.g. subgraphs of a given graph). In accordance with this convention, if G and H are isomorphic graphs we write either $G \cong H$ or simply $G = H$.

Trees: A tree is an acyclic graph.

B.3 Conceptual Data Modelling Techniques

The mathematical basis presented in the preceding section provides a preliminary formalism to define abstract sets, relations, and functions. Modelling the real world requires identifying components, i.e. entities, relationships, and operations, from the real world and representing them with these mathematical constructs. However, the real world is often a complex one in that most of the components identified cannot be directly modelled with basic mathematical tools which are based on homogenous sets. Some rigorous and systematic modelling techniques have to be employed to define and constrain compound structures from basic ones, to practice operations on these structures, and to present the model to others. This section introduces some general concepts on how complex structures can be constructed from simple ones, and then specific modelling techniques for the entity-relationship (ER) data model and the object-oriented (OO) data model will be discussed. These two data models are used, primarily, to describe inter-related concepts from the real world without considering any implementation details with a computer. They are therefore generally called *conceptual data models*.

B.3.1 Modelling Concepts

Classification: The process of classification involves classifying similar objects or concepts into classes. A *class* is analogous to a set. Some traditional set theory claims that a class is too large to be a set. In data modelling, objects in a class correspond to named things in the real world and their internal structures and behaviour often need further definition and explanation using some more primitive concepts. In general, objects of a class have similar

structures and the values of particular properties describing these objects belong to identical attribute domains. However, some of the objects may display properties that make them differ in some aspects from the other objects in the class. These exception objects need to be modelled using additional constraints. *Instantiation* is the inverse of classification and refers to the generation and specific examination of distinct objects of a class. Hence, an object instance is related to its object class by an "IN" or "IS-AN-INSTANCE-OF" relationship.

Observing and classifying similar objects in the real world and putting them into classes is necessary when a bottom-up approach is adopted in data modelling. This process produces abstract data types and allows us to describe or talk about classes rather than the individual objects themselves. Certain properties may apply to the class as a whole and not to the individual objects themselves. For example, the class name and the number of objects in the class are *class properties*. The average value of an attribute over all members of a class is another example of a class property. Incorporated with other modelling concepts, classification can result in fairly complex data types.

Abstraction: One of the main ways of structuring and visualizing data is through the use of abstraction [Tsichritzis and Lochovsky 1982]. Abstraction is the ability to hide detail and concentrate on general, common properties of a set of objects. An elementary form of abstraction distinguishes between the token level and the type level. A *token* is an actual value or a particular instance of an object. Abstraction is used to define a *type* from a class of similar tokens. For instance, abstraction is applied to a set of flowers to form the generic concept FLOWER. In data modelling, abstraction is used to obtain categories of data and combine categories into different levels of more general categories. Data in the same category are supposed to have similarities. Sometimes these similarities are stated as properties of the category. There are two techniques for abstraction: generalization and aggregation [Smith and Smith 1977].

Generalization: This process views a set of tokens or a set of types as one generic type. Token-type generalization is usually differentiated from type-type generalization. The

formal process is referred to as classification, while the latter process is called generalization. For instance, viewing a set of individual river tokens as one generic type RIVER is considered classification. Viewing the types RIVER and LAKE as one generic type WATERBODY is considered generalization. Therefore, generalization is used to describe a phenomenon involving a family of types inheriting some common property which is essential to the phenomenon. It would be more general to use a single concept "waterbody" to refer to the percentage of a surface area covered with water than using concrete concepts such as "river", "lake", "stream", etc.. Analogous to instantiation versus classification, specification is the opposite process to generalization. Thus, a river token is an instantiation of the type RIVER, but the type RIVER is a specialization of the type WATERBODY.

Aggregation: This abstraction structure is composed from its constituent objects. For instance, a city can be characterized by its name, population, and centroid (coordinates of its centre). Hence the attribute types "NAME", "POPULATION", "CENTROID" are aggregated to describe the object type "CITY". The more rigorous use of the aggregation is to relate one object with other higher level objects to form some compound object. To evaluate the land use of a city, for example, it is usual to include objects such as residential areas, commercial areas, parks, etc.. These objects constitute the use of the land within the city and each of them needs finer structures to be completely described. Some of these objects are themselves aggregated from smaller objects and they are essentially different from each other. The reason for bounding them together is because of the specific relation, "land-use", involved. There can be other relationships that bound a city with the objects within it. For example, as parts of the city, residential areas, commercial areas, and parks all "lie in" the vicinity of the city.

B.3.2 The Entity-Relation (ER) Model

The entity-relationship (ER) model (or entity-attribute-relationship model) is one of the best known conceptual models of an information system. It was originally proposed by

Chen [1976] and has been extended with many variations. In the following description, we generally do not distinguish which version of relevant concepts were coined.

The basic components of an ER model, as the name suggested, are: entities, attributes, and relationships. An *entity* may be a person, object, place, concept, or event of interest to be represented in the database. An *entity type* is used to represent a class of similar entities occurring in the real world and is usually identified by a group name. Thus FOREST-STAND, ROAD, RIVER are types of entities. In the type of RIVER, we have an occurrence of a specific river named "Saint Lawrence". An *attribute* (or *attribute type*) is one of the properties identified to describe characteristics of an entity (or entity type). The entity type RIVER, for instance, has one attribute types called NAME, and one called LENGTH, in addition to other possible attribute types. Therefore, the river named "Saint Lawrence" has a specific length of, say 450.0 km.

The ER model features the graphical notation used to represent its components. An entity type can be diagramatically represented by a rectangle box containing the name of an entity type, while its attribute types are represented by ellipses each containing the name of an attribute type. The subordination of an attribute type to an entity type is represented by a line linking a rectangle box and an ellipse. The diagrams of modelling components following this notation are called *entity-relationship* or *ER diagrams*. For example, the ER diagram for the entity type RIVER and its possible attribute types is drawn in Figure B.5. Sometimes attribute types of an entity type can be omitted from the ER diagrams.

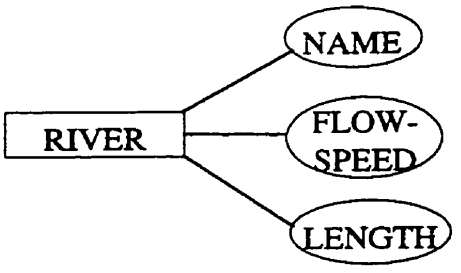


Figure B.5 An entity type and its attribute types

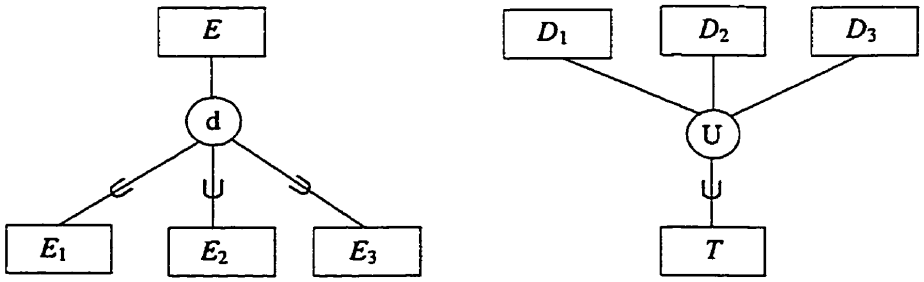
A *relationship type* in the ER model, graphically represented by diamond rectangles, is used to describe an association between one entity type and another. The entity type RIVER and the entity type CITY can be associated with the relationship type PASS-THROUGH. An instance of this relationship finds that Saint Lawrence River passes through the city of Quebec. Sometimes a relationship type can also have attributes which further specify the nature of a relation. Relations can be constrained by cardinalities. A *cardinality* of a relation defines the number of times the relationship can occur between two entities (occurrences). Cardinalities are expressed by four numbers indicating the minimum and maximum numbers of entities occurring in a relation. Figure B.6 illustrates the PASS-THROUGH relation. The cardinality number of the river indicates that a river may pass through zero or n number of cities. Similarly, an occurrence of a city may be passed through by zero or m number of rivers, as indicated by the cardinality numbers by the city. The relation in Figure B.6 is also called many-to-many (or m:n) relationship, as suggested by the maximum cardinality numbers at both sides of the relation. Potentially, one-to-many (or 1:n), one-to-one (or 1:1) relationships can be found in the real world.

Figure B.6 A many-to-many relation involving two entities

The ER model has been extended in various ways to meet practical needs. *The extended entity-relationship* (EER) model adds constructs of more complicated entity types by applying generalization/specification and aggregation techniques described earlier. This leads to the definition of class, superclass, subclass, and category in the EER model. Associated with these concepts is the important mechanism of *attribute inheritance*.

A *class* is a set of entities; this includes any of the EER constructs that group together entities such as entity types, subclasses, superclasses, and categories. An entity type E_1 is a *subclass* of an entity type E if every occurrence of type E_1 is also an occurrence of type E. Accordingly, the type E is said to be the *superclass* of E_1. The superclass/subclass

relationship is often called an *IS-A* relationship because of the way one refers to the concept. A *category* T is a class that is a subset of the union of n defining superclasses D_1, D_2, ···, D_n, $n > 1$. The EER diagrams representing a general superclass/subclass relationship and category are shown in Figure B.7. The subset symbol $\subset$ attached to the linking lines indicates the subclass/superclass relationship. A letter in the circle of each diagram indicates the relationship between classes at the same level, as determined by set operations. For our general example, the letter 'd' in Figure B.7a indicates that E_1, E_2, and E_3 are disjoint in E, while 'U' in Figure B.7b shows that the category T is a subset of the union of D_1, D_2, and D_3.



a. Generating E_1, E_2, E_3 into E

b. Category T is a subset of the union of D_1, D_2, and D_3

Figure B.7 A class/subclass and a category

B.3.3 The Object-Oriented (OO) Model

The most appealing data modelling technology is provided by the paradigm of the object-oriented approach. In the object-oriented approach the concept of object dominates the whole modelling, design, and programming philosophy. An object has not only static structures for its definition, similar to the constructs in the ER model, but also dynamically behavioural ability. The unification of object states (data) and its functions (procedures) in the specification of an object distinguishes the object-oriented approach from those that separate data and the procedures running the data. The object-oriented approach has the following features that are missing from or weakened by a non-object-oriented approach:

Object classes and inheritance: "An *object* is a concept, abstraction, or thing with crisp boundaries and meaning for the problem at hand" [Rumbaugh et al. 1991, pp. 21]. The OO model puts a great deal of emphasis on the design of objects. Objects serve two purposes: they promote understanding of the real world and provide a practical basis for computer implementation. All objects have distinguishing identities which persist through time and are independent of their attribute values. The *identity* of an object may be considered to be represented by a system-generated object identifier (OID). Objects are explicitly created and destroyed. Attributes of an object can be updated without destroying it's identity. "A *class* is a group of objects with similar properties (attributes), common behaviour (operations), common relationships to other objects, and common semantics (the meaning that holds objects together)" [Rumbaugh et al. 1991, pp. 22]. Similar to the description in the ER model, subclasses and superclasses can be defined to represent the static aspects of object structures. The difference with the OO approach is that each class knows how to operate on itself. By declaring a superclass/subclass relationship, the subclass *inherits* all the attributes and operations from the superclass as well as adding it's own. Inheritance makes most of the design and programming effort previously made in the construction of classes to be better utilized.

Methods and polymorphism: "An *operation* is a function or transformation that may be applied to or by objects in a class" [Rumbaugh et al. 1991, pp. 25]. Calculating area, drawing, and colouring the interior are operations on class AREAL-OBJECT. All objects in a class share the same operations. In the OO paradigm, a *method* is the implementation of an operation for a class. The same operation can be implemented in different classes with different procedures. This phenomenon is called *polymorphism*. For example, the class AREAL-OBJECT may have RECTANGLE, CIRCLE, and TRIANGLE as its subclasses. All these subclasses inherit the method "calculate-area". The codes implemented in each subclass, however, are different. Polymorphism becomes powerful in combination with inheritance. It provides flexibility for execution of processes in information systems, because operations need only be bound to implementations at run-time.

Encapsulation: *Encapsulation* (also referred to as *information hiding*) consists of separating the external aspects of an object, which are accessible to other objects, from the internal implementation details of the object, which are hidden from other objects. Encapsulated objects or components can be used and reused in developing computerized systems, which prevents a system from becoming so interdependent that a small change has massive ripple effects. The implementation of an object can be changed without affecting the applications that use it [Rumbaugh et al. 1991, pp. 7].

With the OO approach, an object knows how to act itself. To activate an object to invoke its methods, however, needs messages communicated from another object. External stimuli are generated by events which occur when objects in a system participate in activities. Therefore, from the system's point of view, a data model should be equipped with an environment in which processors (embedded in objects), corresponding to system functions, can be managed, coordinated, and controlled. The environment serves as an interface exposed to objects. The public interface of an object contains a collection of messages to which the object responds by altering its state or returning an object. The interpretation of the messages passing between objects shows what happened to the objects in a system. These concepts address three important aspects of an information system based on the OO data model: "the *object model* represents the static, structural, 'data' aspects of a system; the *dynamic model* represents the temporal, behavioural, 'control' aspects of a system; and the functional model represents the transformational, 'function' aspects of a system" [Rumbaugh et al. 1991, pp. 17]. Modelling these aspects is the content of a particular methodology named the Object Modelling Technique (OMT) [Rumbaugh et al. 1991]. The philosophy, concepts, and notations will be applied in this thesis for the design of applications using our spatial data model.

B.4 Database Modelling and Design Process

In this section, we discuss commonly practiced modelling processes for the design of applicational database systems. It turns out that different phases of data modelling exist

from the real world to the computer implementation. Each phase is characterized by a set of distinct modelling techniques and produces a mapped state of the real world in accord with the underlining technology (Figure B.8). Of the five phases identified, phase 1 is concerned with analyzing potential application areas; identifying data classes and attributes and functions to provide services and products; and constraints over data and functions within an organization. Approaches in this phase include examining documents and interviewing users at different levels of the organization. Results from this stage may include data classes and data-flow diagrams, a matrix showing which functions use which data classes, and metadata documentation, etc. Phase 5 concerns populating a database with data and coding with a database language specified from previous phases. "These two extremes are usually not considered as part of the database design, but part of the more general information life cycle" [Elmasri and Navathe 1989 pp. 458]. The heart of the database modelling and design process is phases 2, 3, and 4 which are discussed below:

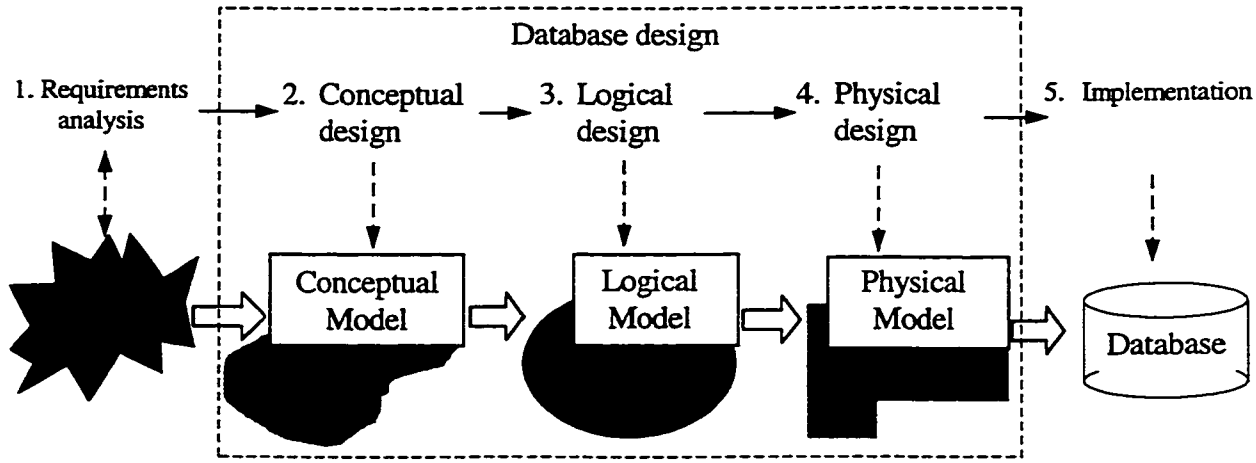


Figure B.8 Conceptual, logical, and physical design and models

Phase 2: Conceptual design. This phase "produces a conceptual schema for the database that is independent of a specific DBMS" [Elmasri and Navathe 1989 pp. 458]. A conceptual schema is a formal understanding of the database structure, meaning (schematics), interrelationships, and constraints. A high-level data modelling technique (such as the ER model and the OO model) can be used to achieve a formalized description of the database. All concepts and relationships identified during conceptual modelling constitute the explicit

information contents of a database. The presentation of this information is often called a *conceptual model.* A good conceptual model should also allow flexible manipulation of concepts and relationships to derive new concepts and relations. Conceptual modelling plays a paramount role in designing an information system. It influences the expressive power and operational capability of the information system build from it. The conceptual model is, however, most general and is independent of any software or hardware. The conceptual model should possess the following characteristics [Elmasri and Navathe 1989 pp. 462]:

1. Expressiveness: The data model should be expressive enough to point out commonly occurring distinctions between different types of data, relationships, and constraints.
2. Simplicity: The model should be simple enough for typical users to understand and use; its concepts should be easily understood by end users.
3. Minimal: The model should have a small number of basic concepts that are distinct and nonoverlapping in meaning.
4. Diagrammatic representation: The model should have a diagrammatic notation for displaying a conceptual schema that is easy to interpret.
5. Formality: A conceptual schema expressed in the data model must represent a formal nonambiguous specification of the data. Hence, the model concepts must be accurately and unambiguously defined.

A conceptual model can be obtained by an incremental approach. One starts with a preliminary schema constructed from the previous phase, and incrementally modifies and refines it. For large databases it is sometimes difficult to try to design the whole database schema at once. In such cases, individual views (small schema) can be designed first by using a bottom-up, top-down, or mixed strategy and then integrated [Elmasri and Navathe 1989 pp. 463-464].

Phase 3: Logical design. The next step after conceptual modelling transforms a conceptual schema into the schema of a particular DBMS. A DBMS schema consists of structures, constraints, and languages. A collection of structures specifies data and attribute types, and

relationships a DBMS schema supports. The set of constraints is used to ensure the integration of schema structures, which is done by examining functional dependencies of the structural components. The languages provide a host of vocabularies and syntax that one can use to define allowable structures (called data definition language DDL) and dynamic actions to manipulate data and structures (called data manipulation language DML).

It is apparent that a DBMS schema has an inherent data model based on which the DBMS is built. The basis of a DBMS data model encompasses some mathematical theory and structures which operate on more or less homogenous entity and attribute domains (abstract sets) and generates mathematical constructs for complex relationships. Concepts and relationships from a conceptual model sometimes need to be broken into homogenous components at this level. The relationships between components are maintained by logical links provided by the DBMS schema. For this reason, the representation produced by following a DBMS schema is referred to as a *logical model.* Except for mathematical concepts used to capture and manipulate relationships, a logical model does not directly involve any computing data structures. In fact, it hides the actual data structure from the user. The way of constructing a logical data model, however, affects data structures chosen to implement a system.

Since the late 1960s, the development of DBMS data models has experienced several evolutions. The earliest data models, such as the network and hierarchical models, were drawn from early file processing and report generation systems [Fry and Sibley 1976]. They have gradually been replaced by a more popular data model, the relational model first proposed by Codd [1970]. A significant reason for the popularity of relational data models is that they are simple in nature and are strongly based on the mathematical theory of relations.

The only data structuring tools used by original relational data models is a relation. The definition of a relation in relational models is identical to the mathematical one except that database relations are time varying. That is, tuples are inserted, deleted, and modified in

database relations. The definition of a database relation [Codd 1970] is: given sets D_1, D_2, ···, D_n (not necessarily distinct), R is a relation on these n sets if it is a set of *n-tuples* or simply *tuples* each of which has its first element from D_1, second element from D_2, and so on. The sets D_i are called the *domains* of R. The number n is the *degree* of R, and the number of tuples in R is called its *cardinality*.

From the basic definition, a relational database can be specified by a *relational schema* which consists of one or more *relation schemes*. A relation scheme is a listing of a relation name and its corresponding attribute names. In a relational model, relation schemes are represented as named *tables* with rows being tuples and columns being named *attributes*. An instance of a relation scheme, i.e. a *relation*, is a finite set of tuples each containing as many data items as there are attribute names in the relation scheme. Each data item has a value from the domain with which its attribute type is associated. A relation has the following properties:

- The ordering of tuples in the relation is not significant;
- Tuples in a relation are all distinct from one another; and
- Columns are ordered so that data items correspond to the attribute in the relation scheme with which they are named.

Traditional set operations, such as union, intersection, and difference; together with relational ones, such as project, join, and divide; are supported in a relational database and are the basis of the so called *relational algebra*. Constraints applied to a relational schema are guided according to the three directions: *representation*, *nonredundancy*, and *separation*. They constitute important contents of the schema analysis or formalization [Tsichritzis and Lochovsky 1982] which are used to obtain a good schema. Data definition and manipulation with a relational database are facilitated by the structured query language (SQL) which provides a standardized syntax and semantics for users to define relation schemes and then insert, modify, and retrieve data from a relational database.

Phase 4: Physical design. This is the process of choosing specific storage structures and access paths for the database files to achieve good performance for the various database applications. The design of physical data structures is concerned with various type of indexing, clustering and linking of related records on disk blocks via pointers, hashing keys, and so on. Criteria for the choice of physical data structures may include space utilization for data and their structures, and response time needed for execution of transactions.

The whole data modelling process involves transforming one data model to another. It is possible that not all information from a source model may be preserved or realized efficiently in a target model. The information loss between representations of different data models is termed *impedance mismatch* [Worboys 1995, pp. 85, 93], which can be aggravated if the chosen data models do not have a high compatibility in data and relationship types.