

Portland State University

**PDXScholar**

---

Dissertations and Theses

Dissertations and Theses

---

1993

# Object Parallel Spatio-Temporal Analysis and Modeling System

David Bruce Rex

*Portland State University*

Follow this and additional works at: [https://pdxscholar.library.pdx.edu/open\\_access\\_etds](https://pdxscholar.library.pdx.edu/open_access_etds)

**Let us know how access to this document benefits you.**

---

## Recommended Citation

Rex, David Bruce, "Object Parallel Spatio-Temporal Analysis and Modeling System" (1993). *Dissertations and Theses*. Paper 1278.

<https://doi.org/10.15760/etd.1277>

This Dissertation is brought to you for free and open access. It has been accepted for inclusion in Dissertations and Theses by an authorized administrator of PDXScholar. Please contact us if we can make this document more accessible: [pdxscholar@pdx.edu](mailto:pdxscholar@pdx.edu).

OBJECT PARALLEL SPATIO-TEMPORAL ANALYSIS  
AND MODELING SYSTEM

by

DAVID BRUCE REX

A dissertation submitted in partial fulfillment of the  
requirements for the degree of

DOCTOR OF PHILOSOPHY  
in  
URBAN STUDIES

Portland State University  
© 1993

---

TO THE OFFICE OF GRADUATE STUDIES:

The members of the Committee approve the dissertation of David Bruce Rex  
presented June 22, 1993.

  
Kenneth Dueker, Chair

  
Richard Lycan

  
William A. Rabiega

  
James Strathman

  
Roy W. Koch

APPROVED: 


  
Nohad A. Toulan, Dean of the School of Urban and Public Affairs


  
Roy W. Koch, Vice Provost for Graduate Studies and Research


AN ABSTRACT OF THE DISSERTATION OF David Bruce Rex for the Doctor of  
Philosophy in Urban Studies presented June 22, 1993.

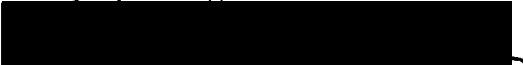
Title: Object Parallel Spatio-Temporal Analysis and Modeling System

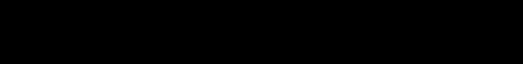
APPROVED BY THE MEMBERS OF THE DISSERTATION COMMITTEE:

  
Kenneth Dueker, Chair

  
Richard Lycan

  
William Rabiega

  
James Strathman

  
Roy Koch

This dissertation will outline an object-oriented model from which a next-generation GIS can be derived. The requirements for a spatial information analysis and modeling system can be broken into three primary functional classes: data management (data classification and access), analysis (modeling, optimization, and simulation) and visualization (display of data). These three functional classes can be considered as the primary colors of the spectrum from which the different shades of spatial analysis are composed. Object classes will be developed which will be designed to manipulate the three primary functions as required by the user and the data.

THIS DISSERTATION IS DEDICATED TO  
THE MEMORY OF

SUSAN T. RHYNEER

## TABLE OF CONTENTS

	PAGE
LIST OF FIGURES .....	vi
CHAPTER	
I OVERVIEW AND INTRODUCTION .....	1
Overview .....	1
Introduction .....	3
The Current Generation	
The Next Generation	
The Starting Point	
General Requirements - Problem Definition and Objectives .....	9
Assumptions	
Alternatives	
Constraints	
Evaluation of Alternatives with Respect to Constraints	
Language, Platform and Operating System	
II THE OBJECT PARADIGM .....	13
Background of the Object Paradigm in GIS .....	13
Introduction to Object Orientation	
Simple Objects	
Inheritance	
Multiple Inheritance	
Encapsulation	
Polymorphism	
Binding	
Container Classes	
Complex Objects	

<b>III</b>	<b>PARALELLISM .....</b>	<b>21</b>
	Parallel Architectures .....	21
	SIMD	
	MIMD	
	Decomposition .....	25
	Perfectly Parallel Decomposition	
	Domain Decomposition	
	Control Decomposition	
	Layered Decomposition	
	Parallelism for Neural Networks	
	The Appropriate Parallel Architecture for GIS	
	Object-Parallelism	
	Parallel Objects	
<b>IV</b>	<b>TOWARD AN OBJECT-ORIENTED DATA MODEL .....</b>	<b>32</b>
	Overview .....	32
	Basic Object-Oriented Models	
	Knowledge-based Models	
	Spatio-Temporal Models	
<b>V</b>	<b>THE OPSTAMS DATA MODEL .....</b>	<b>50</b>
	Model Overview .....	50
	Computational Geometry and k-Dimensionality	
	Visualization of Temporal Data in the OPSTAMS Environment	
	Selection Four-Dimensional Data from a Two-Dimensional Display	
	Class Structures	
	The Geometry Class	
	The Data Management Class	
	The Analysis Class	
	The Display Class	
	Polymorphic Functions	
	Decomposition	
	System Level Functionality	
	A User's Perspective	

<b>VI</b>	<b>ARTIFICIAL NEURAL NETWORKS IN SPATIAL ANALYSIS .....</b>	<b>70</b>
	Introduction to Artificial Neural Networks .....	70
	Discriminants and Hyperplanes	
	Learning	
	Neural Models	
	Neural Nets and the Object Paradigm	
	Neural Nets and Parallelism	
	Neural Nets in Spatial Analysis	
<b>VII</b>	<b>THE OPSTAMS MODEL APPLIED TO SHORTEST PATH PROBLEMS</b>	<b>80</b>
	Overview .....	80
	The Dijkstra Algorithm	
	The Tsai and Ma Algorithm	
	Summary, Conclusions and Suggestions for Future Work .....	94
	Advantages of the OPSTAMS Model	
	Disadvantages of the OPSTAMS Model	
	Other Implementation Issues	
	Evaluation	
	<b>SELECTED BIBLIOGRAPHY .....</b>	<b>103</b>
	<b>APPENDIX .....</b>	<b>111</b>



## LIST OF FIGURES

FIGURE		PAGE
1.	Functionality of Current vs. OPSTAMS-based Systems	12
2.	Object and Class Notation	15
3.	Use of Point Objects in the Line Class	16
4.	Single Inheritance	16
5.	Multiple Inheritance	17
6.	Complex Objects	19
7.	Class Diagram of a Complex Object	20
8.	SIMD Architecture	22
9.	MIMD Architecture	23
10.	The Z-M Class Hierarchy	35
11.	The Z-M Model Relationship Hierarchy	36
12.	A Drainage Basin	37
13.	The M-R-B Model Representation of a Drainage Basin	38
14.	A Road and Forest at Time $t_0$ (a) and at Time $t_1$ (b)	42
15.	Combined Geometry of Two Time States	43
16.	Cell Complexes	45
17.	Spatio-Temporal Cell Complexes	46
18.	Spatio-Temporal Simplices and Complexes	48
19.	The Hierarchy of ST Objects	49
20.	Granularity of Time	52
21.	The Fundamental Abstract OPSTAMS Classes	53

22.	A Class Member Function	55
23.	The Line Class Inherits from the Vector Abstract Class	55
24.	The Display Class	59
25.	Many-to-Many Relationships Between Classes	64
26.	The Rider Class	65
27.	Process Flow Diagram for the Rider Example	68
28.	A Typical Neural Network Topology	71
29.	Neuron Activation	71
30.	Discriminants	73
31.	The Topology of a Hopfield Net	76
32.	Class Structure for Object Dijkstra	82
33.	Class Containment in Object Dijkstra	83
34.	A MIMD Topology	87
35.	Tsai and Ma Class Containment	91

## CHAPTER I

### OVERVIEW AND INTRODUCTION

#### OVERVIEW

The primary problems this dissertation will address are 1) that the current generation GIS, which is based on relational data models, does not provide optimal problem abstraction to a system in that it does not adequately address the needs of analysts with respect to three spatial dimensions and time (four-dimensional data), and 2) is not designed to take advantage of the coming shift from sequential computers to computer architectures based on massive parallelism.

The dissertation will outline a model for and demonstrate the applications and advantages of a next generation spatial information analysis system. The first part of the thesis will provide an overview of the system and will discuss the concept and its rationale in greater detail. The second part will introduce the object-oriented paradigm and will explain a data structure for spatial analysis in great detail. The thesis will suggest that these object-oriented data structures will give the system tremendous power of abstraction for the analyst [Zhe93]. The supporting data structures provide a foundation for a logical data model.

The third section will explore parallelism and will illustrate why it is so important to future systems. Neural network algorithms, algorithms from computational geometry and operations research will receive special attention. Algorithms from these disciplines can be combined to form analysis objects which can be linked together in support of individual problem solutions.

Later sections will present an approach to modeling spatial relationships in four dimensions. The model presented is developed from the foundation introduced in the second section. Visualization of spatial data in time will be addressed and a methodology introduced. Additionally, a spatial analysis problem will be developed using a next generation model and offer a solution technique based on the application of relevant algorithms. The problem which will be developed will be an optimal vehicle routing on a topologically dynamic network.

## INTRODUCTION

Now that GIS has become relatively mature as a methodology for dealing with spatial data, many large data sets have become available for mapping many possible surface entities. There are parcel basemaps, TIGER files, Digital Line Graphs (DLG), Digital Elevation Models (DEM), Digital Terrain Models (DTM), utility basemaps and thematic maps of all sorts. Large data sets and complicated analyses are pushing the current generation of GIS to its limits. This tremendous availability of spatial data is exciting, but the current generation of GIS is falling short in its ability to both manage and process this plethora of data [Gah88]: The current generation falls short in the ability to perform operations on large spatial data sets and to provide adequate visualization of the temporal component. These failings are primarily due to the fact that the current generation of GIS is based on twenty-year old technology.

As the current toolset becomes ever more dated, more universities are offering GIS-related degrees, which has resulted in a new generation of more sophisticated spatial analysts. This new generation of analysts will require more analytic power from future systems. People tend to solve problems in terms of the toolset with which they are familiar. Given an expanded toolset, analysts will be able to conceptualize problems in terms of these new capabilities. Thus, the demand for analytic power will push system designers to provide even more advanced analytic capabilities. And so the cycle goes.

Although any GIS' ability to handle large data sets is constrained to some degree by the throughput capacity of its host hardware's storage device, an inherent constraint exists due to the relational data model underlying today's systems. The relational model forces cumbersome two-into-one table join operations on the analyst whenever he or she

performs an analysis that results in a new layer or map. Similarly, relational models have been unable to adequately address the temporal component of spatial in any manner other than "snapshots", which are typically stored as tables of geometry for different slices of time. The relational models are unable to deal with time as a continuous function, only as discrete instances. This approach has proved to be inefficient from both storage and analytic perspectives.

In addition to the standard spatial-analytic functionality offered by the current generation of systems such as spatial overlay, intersection and proximity analysis, new systems will need to be able to address several new classes of problems related to visualization, image processing, and combinatorial optimization posed by increased attention to four-dimensional data, due to both the increasing availability of such data and an increased analytic acumen of new graduate analysts.

The proper visualization of data containing a strong temporal component requires animation. The current generation of single processor computers is presently incapable of animating anything more than the most simple of data sets in real time. Image processing techniques frequently utilize complicated algorithms for improving the image quality of a single image. When many frames of images are involved, as in the animation of images of a spatial entity over time, current single processor systems are simply too slow. Parallel computing platforms provide tremendous computational potential over single-processor-based platforms.

Pattern recognition in remotely-sensed data can benefit from artificial neural network-based analytic techniques made available by parallel computers. Many optimization problems that require heuristics to return a close approximation to global optima because of the performance constraints of single processor computers can be replaced by exhaustive search algorithms run on parallel processors in short periods of time which return exact solutions.

Appropriate techniques for dealing with new problem classes must also be available to the analyst. Analytic techniques based on computational geometry and artificial neural networks will play an ever increasingly important role in the processing and analysis of spatial data. These techniques will both benefit and rely upon the object-oriented paradigm and parallel computer architectures for easing problem abstraction and providing the level of performance required for the analysis of large data sets. Thus, an object/parallel approach to system design is an alternative to the relational/sequential model used by the current generation of systems.

The relational model is constrained by the inability to adequately model time in relational databases and sequentially programmed applications are limited by the clock speed of the single processor. The object/parallel approach has the potential to overcome these two vital shortcomings of the relational/sequential model, on which the current generation is based.

### The Current Generation

Up to now, most of the effort in GIS has been directed toward loading the systems with data (storage) and displaying that information accordingly (retrieval). The power of this information comes with our ability to utilize it, or more specifically, to analyze it. The current generation of GIS has evolved from a display-oriented set of development criteria, namely thematic mapping.

The analysis tools that are available in the current generation of GIS have been added on to the map-oriented systems almost as an afterthought, so as to make the systems a little more useful to moderately sophisticated analysts. As Oliver [Oli90] states, "...the digital mapping era of GIS is being transcended by the demand for more specific and advanced forms of geographical analysis. The next advance must be the incorporation of suitable analytical procedures that can be linked with the existing facilities

to display data".

### The Next Generation

In order to get the most from our data, the next generation GIS must be developed on an analysis-oriented basis. Just as the new method of teaching mathematics employs the use of calculators and small computers to perform the rote tasks of arithmetic, algebra and even calculus, so that students may concentrate on the formulation and abstraction of problems, students of geography, urban studies and other related disciplines will use GIS for fundamental information gathering and display, and then be able to concentrate on problem abstraction and analytic methodology. As our abilities to analyze become more sophisticated, so will the need for more sophisticated geographic information (spatial) analysis systems [Ope88], [Arm90-1].

These systems will be highly integrated [Cho92] and have a set of primitives in the forms of data structures and algorithms which will be used to construct models for performing spatial analysis [Gah88.]. They will take advantage of the latest trends in software design and computer architecture in the forms of object-orientation (O-O) and distributed processing (parallelism). They will be knowledge- or rule-based "expert" systems to some degree. They will be capable of four-dimensional analysis (3-D Euclidean space plus time)[Haz90]. Such will be the foundation upon which applications will be built that can effectively model large and fine-grained problems rooted in any field utilizing spatial analysis [Goo91].

### The Starting Point

The current generation of GIS is just barely beginning to venture into the arenas of object-oriented data structures and parallel computer architectures. Examples of these applications are the Intergraph's TIGRIS and Computervision's System 9 which utilizes an



object-oriented data structure for the spatial data model coupled with the Empress RDBMS, and the ARC/Oracle/KIVA/Sequent/Sun link, where the attribute data stored in the Oracle relational database is passed to a Sequent parallel processing computer for operations generated from database queries.<sup>1</sup>

Both of the aforementioned systems are traditional GIS, offering the standard functionality. The latter example does not utilize the parallel processor for any of the spatial-analytic functions of ARC/INFO - these are still sequentially processed on the Sun Sparcstation as are all the screen graphics operations. The former's object-oriented spatial data model is mitigated by its mapping to a relational data storage paradigm.

The use of a PC-based transputer net has been suggested [Hea90] and recently applied to the shortest paths problem [Din92]. While there have been many recent additions to the computer science literature on parallel shortest path algorithms, the work by Ding, et.al. [Din92] is the first to appear in the GIS literature. While transputers are inherently capacitated by PC-based operating systems, this work is encouraging.

Another issue which is currently a hot topic in the literature but has not adequately been represented in the marketplace is the issue of time in spatial analysis [Pri89] and [Lan89]. The new generation of systems will need to include data structures which can adequately handle spatio-temporal data within the kernel data structure [Pig92], [Smi92] and [Wor92]. The thesis developed here will provide a methodology for implementing spatio-temporality in future systems.

This next generation of spatial analysis systems will need to be designed from a fresh slate, as opposed to hanging more bells and whistles onto the current generation's fundamental architecture. The next generation of analysis tools need to be designed around two primary objectives: 1) the analyst must be able to abstract a problem in "real

---

<sup>1</sup> Silicon Graphics Corp. has a SIMD-based image processing system available, but does not currently offer a parallel GIS.

world" terms and 2) solve the problem in "real world" time [Bru92]. This notion of "ease of abstraction" is paramount to the approach presented here.

Two new architectures must be utilized to implement these objectives. On the hardware side, a parallel architecture will be required to provide the necessary rapid processing needed for large data sets and CPU-intensive processes. On the software side, the object-oriented paradigm of software and database design will be used to provide better support for problem abstraction, the system's code development and user interface.

A great deal of parallel-based software design (and re-design) is certainly necessary as well, in that our current set of algorithms in spatial analysis are sequential by nature, so they must be "parallelized" in order to function properly in a distributed environment. New algorithms will need to be designed from scratch in many cases. There is currently much activity in this area [Hop92], [Din92], etc. The utilization of parallel computing architectures also permits artificial neural network algorithms to be easily applied to spatial-analytic problems. It seems appropriate to label this Object-oriented, Parallel-based Spatio-Temporal Analysis and Modeling System "OPSTAMS", and this will be the acronym used for this future system henceforth.

## GENERAL REQUIREMENTS - PROBLEM DEFINITION AND OBJECTIVES

As stated above, the next generation systems must offer ease of abstraction with respect to problem space. They must offer the ability to visualize data in the three dimensions of space, and time. They must offer this functionality in a timely manner. As data sets become larger and analysis becomes more complicated, performance becomes a critical factor in the efficacy of analysis systems. The methodology presented here supports these objectives.

The objective of this dissertation is to present a new model that 1) eliminates the constraints imposed by the relational model through the application of object orientation and 2) is viable with respect to the new parallel architectures of computer hardware.

The success of an implementation of these concepts can be measured against current generation by 1) the ability of the system to model temporal data, 2) the ability of the system to analyze and manipulate large data sets with a minimum of operational steps (eliminate multiple relational join operations), and 3) the ability of the system to operate on a parallel computing platform.

### Assumptions

The underlying assumptions to the OPSTAMS approach are that the current generation of system design is growing inadequate and the approach presented here is currently implementible. Although expensive, hardware does exist that would allow a system designed around the OPSTAMS model to execute. Combinations of the "C" and "C++" will allow such a system to be developed.

### Alternatives

The alternatives presented to the developer attempting to address the shortcomings of the current generation of systems are 1) to adapt traditional technologies (computer languages and computing platforms) to the new paradigm, 2) introduce new technologies to work with the new paradigm, or 3) to use traditional technology where appropriate and introduce new technologies when needed. The model presented here is based on the third alternative.

### Constraints

The primary constraints relevant to the development of a new spatio-temporal modeling and information system paradigm are 1) constantly evolving computing architectures (hardware), 2) the evolution of new operating systems in support of these new computing platforms, and 3) the constant evolution of computer languages.

### Evaluation of Alternatives with Respect to Constraints

In order to be successful with the introduction of a new paradigm, its introduction must be farsighted enough to be immediately implementible and usable, as well as being adaptable and extensible so that it is persistent into the near future (five to ten years). To support these objectives the OPSTAMS approach takes the path of utilizing traditional technologies with new technologies as necessary.

### Language, Platform and Operating System

Given this new paradigm needs to be implemented via an object-oriented language, the set of choices for such a language is quite small. Basic, Fortran and Pascal are not inherently object-oriented. To attempt to implement a new, sophisticated system in one of these languages would be fraught with ungainly workarounds to achieve "object-orientedness". Although not object-oriented, there are many implementations of Fortran

on multiprocessor platforms.

Although many object-oriented languages exist such as Smalltalk, Actor, Eiffel, Ada and Lisp, none of these languages has emerged as the de facto standard object-oriented language and many are only supported on one or only a few computing platforms. Additionally, little work has been done in porting these languages to platforms capable of parallel computation.

C++ has become the de facto object-oriented language standard and has been successfully ported to multiple processor platforms. C++ offers the additional power of being backwardly compatible with the C language, so that legacy code used to develop analytic models can be incorporated into new systems. Minor extensions to the language are all that is needed to support parallelism and several currently exist. Most importantly, because C++ has evolved into the new standard, it is being supported by all major computing platforms (including those offering parallelism), and is the most portable. Thus, C++ is currently the best choice on which to base the design of a new approach to spatio-temporal modeling and information system.

Figure 1 provides an overview of the various functionality provided by the current generation, those supported by the OPSTAMS model and a relative measure of change offered by a system based on the OPSTAMS model in comparison to systems of the current generation.

Operation or functionality	Support by current generation	Support by OPSTAMS	Expected degree of improvement
<b>Data Management</b>			
Data import/ export formats	Good	Yes	Low
Data transformation	Fair	Yes	Moderate
Projection transformation	Good	Potential	Moderate
I/O from storage device	Fair	Yes	Moderate
Object-oriented data organization	Poor	Yes	High
Relational data organization	Good	No	None
<b>Geometry</b>			
2-D Vector spatial data model	Good	Yes	Low
3-D Vector spatial data model	Low	Yes	Moderate
Raster spatial data model	Good	Yes	Low
Temporal data model	Poor	Yes	High
Integrated raster/ vector model	Fair	Yes	High
N-space hyperplanes	None	Yes	Extreme
<b>Analysis</b>			
Intersection/ Overlay analysis	Fair*	Yes	High
Proximity analysis	Good	Yes	Moderate
Buffer analysis	Good	Yes	Moderate
Complex feature construction	Fair*	Yes	High
Graph theory-based analysis	Good	Yes	Moderate
Multidimensional analysis	Poor	Yes	High
Neural network analysis	None	Yes	Extreme
Parallel algorithms	None	Yes	Extreme
Temporal analysis	Poor	Yes	High
<b>Display</b>			
Animation of temporal data	Poor	Yes	High
Volumetric rendering	Fair	Yes	High
Cartographic output	Good	Potential	Low
* constrained by relational joins			

Figure 1. Functionality of Current Systems vs. OPSTAMS-based Systems

## CHAPTER II

### THE OBJECT PARADIGM

#### BACKGROUND OF THE OBJECT PARADIGM IN GIS

Over the past several years, the term "object-oriented" has become quite the buzzword in the GIS community. The object-oriented paradigm has been applied to spatial data models since 1986 [Kje86] [Kje88], but at this time, only two vendors have come forth with a true object-oriented spatial data model for a GIS. Object-oriented programming allows both data and procedures to be combined (encapsulated) together as a unique object.

Objects are organized into taxonomic structures referred to as class hierarchies. Class hierarchies can be defined by data, procedures (methods), or by a hybrid rule of data type and method. By properly organizing and defining the relationships between classes, the real world may be more accurately abstracted into computing data structures. For example, an object will inherit attributes from a superclass and, in turn, pass on attributes to a subclass. In this manner one can move up and down through the class structure adding members to objects as necessary to achieve accurate and complete representations of real world entities (generalization and aggregation, [Ege90]).

The data models employed by Computervision's System 9 and Intergraph's TIGRIS are O-O due to inheritance, but do not support a macro language based on the object structure which can utilize encapsulation (combining objects and procedures together) or polymorphism (context-sensitive procedures sharing the same name) so that the analyst is able to work with the spatial data in a purely O-O environment.

Recently, the literature has proffered integrated expert system approaches to the incorporation of the O-O paradigm into GIS [Gah88] [Arm89] [Arm90] [Wor90] and [Oat90]. It has been shown that O-O abstraction is useful for real world modelling and when dealing with very large data sets [Gah88]. Armstrong [Arm89] provides an O-O approach to location-allocation modelling based on the Smalltalk language and the cadastral model of Kjerne and Dueker [Kje88].

Most researchers in this field to date have attempted to devise an object-oriented spatial data structure and then "float" the structure over traditional entity-relationship based database management systems such as INGRES or ORACLE. This is a step in the right direction, but a true object-oriented information system must be designed from the ground up in order to function properly as such. Anything less will be a kludge (which Webster defines as "...a computer system made up of poorly matched components").

### Introduction to Object Orientation

In object oriented programming, there are five fundamental concepts which give programmers the power to effectively model complex real-world situations. These five concepts are simple objects (sometimes referred to as abstractions [Zha92]), inheritance, encapsulation, polymorphism and binding. The fifth concept, binding, is compiler-related but allows polymorphism to be used to full advantage. Used together, these concepts can be applied to almost any problem which is inherently hierarchical in order to organize data and procedures into an object oriented model (program).

### Simple Objects

A simple object can be data, a behavior (procedure) or a combination of the two (more on that later). Objects are the primary level of abstraction. For instance, a point somewhere on the Earth might be abstracted as "Point" identified by "ID", "Latitude" and



"Longitude". In this case, the object is comprised of data. Another example of an object is a procedure called "Sort" which takes a list of x-y coordinates and sorts them on x.<sup>1</sup> Classes of objects are comprised of data and behaviors. An object is an instance of the class (figure 2).



Figure 2. Object and class notation.

### Inheritance

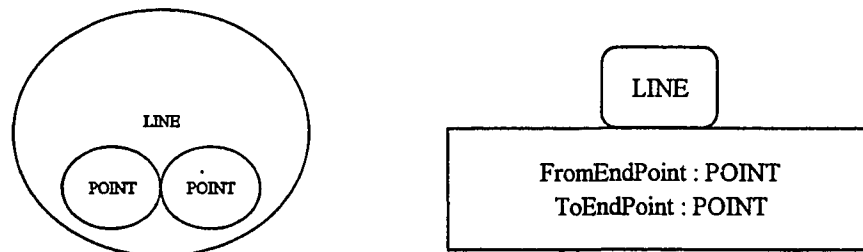
An object can share data or behavior with other objects. Objects are organized according to their commonalities with respect to data and/or behavior. These organizations are referred to as "classes". Objects are referred to as *instances* of classes. There are base classes and derived classes of objects. Some authors refer to the concept of inheritance as "generalization" with base classes as "super" or "parent" classes and derived classes as "sub" or "child" classes. The concept is the same, regardless of the terminology employed. This dissertation will use "base" and "derived" to avoid the confusion that sometimes surrounds the super- and subclass terminology [Lad90]. For example, we can create an instance of a class called "Point", with ID of 1, Latitude of 36.1254 and Longitude of -107.2341, which is now our base class.

Our point is an instance of class Point. Suppose we have another object called "Line". A line is comprised of two points (figure 3). Next we create a Line class. An instance of Line class inherits point characteristics from the Point class. Line then contains

---

<sup>1</sup> See Appendix A regarding notation.

an ID for the line, a FromEndPoint and a ToEndPoint. The From- and To- endpoints inherit Point ID, Latitude and Longitude from class Point.



**Figure 3.** Use of Point objects in the Line class.

Line may be a derived class of Point or it may simply "use" points and not inherit from the Point class. Whether or not a class inherits or uses is up to the programmer/designer. In figure 4a, Point is a base class of Line (Line derives from Point). Throughout the class hierarchy, a class may be derived from one class and at the same time be a base class for another (figure 4b). It is important for the programmer or analyst to carefully prethink class structures before committing them to code. Poorly conceived class structures may be cumbersome and defeat the advantages which can be gained from the object oriented approach.



**Figure 4.** Single inheritance.

### Multiple Inheritance

Classes derived from more than one base class are considered to have multiple inheritance. A new class can be comprised of attributes from one base class and methods (behaviors) from another. For example, if we had a class called "Util" which contained a method called "Sort()", and combined "Util" with "Line" to form a new class "LineUtil", a LineUtil object could sort the coordinates of a given line (figure 5). Any number of objects from different base classes can be combined to form a new derived class. This is essential for the proper characterization of complex objects.

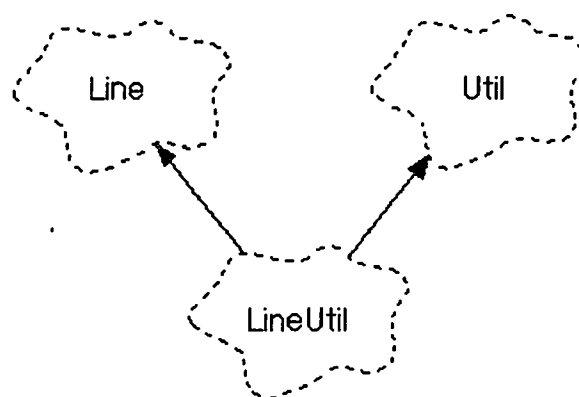


Figure 5. Multiple inheritance.

### Encapsulation

Earlier we created two classes, each comprised of data. The Point class has ID, Latitude and Longitude. The Line class has an ID, a FromEndPoint and a ToEndPoint. The concept of encapsulation allows us to add a method to the class which can act on the data residing within the class. The procedural analog to a method is a routine. For example, a method "Length()" would calculate the straight-line distance between the endpoints of an instance of class Line. Thus the Line class contains both data (lines, which are instances of the class) and methods (the Length() routine). Subclasses of the Line

class will be able to inherit the method Length() from Line.

### Polymorphism

A method which can be applied to many classes is considered polymorphic. We have created the Point and Line classes, but have no way of displaying them. What is needed is a routine that will draw a point or a line on the computer screen. A method Draw() can be created which will paint a pixel at the screen coordinates determined by an instance of Point class' Longitude and Latitude. A similar method can be created for Line and can also be called Draw(). Thus, the Draw() method can be called when a point or a line is to be painted to the screen and the Draw() method from the appropriate class will be selected.

### Binding

The means by which the proper Draw() method is selected is referred to as *binding*. In C++ binding can be early or late. In early binding, the proper Draw() method is matched to the appropriate data type (determined by class) at compile time. Late binding chooses the appropriate method at run time. Late binding produces larger executable code sizes and can slow program execution, but can reduce the total lines of code which need to be written in order to cover all possible uses of Draw(), for example. Additionally, an important advantage of this approach is that users of the developed program can benefit from deriving new classes from existing base classes, thereby reducing redundant code as might be the case from a procedural approach. This lends flexibility to the resulting system and aids the analyst in abstracting the problem to the system.

### Container Classes

A container class is a class which defines objects that contain other "sub"objects

and methods. Container classes may be constructed in order to solve a particular problem by combining specific objects with specific methods, or may be created as a convenient place to keep certain types of routines. In the latter case, these classes are sometimes referred to as "bags". Container classes are always derived classes and can take full advantage of multiple inheritance, encapsulation and polymorphism. For example, the `LineUtil` class constructed above is an example of a container class. Other methods for operating on data structures such as `CreateList()` and `Insert()` might be included from a class called `LinkedList`. These objects and methods can then be combined and used together to create and manage a linked list of lines or a list of endpoints of the lines.

### Complex Objects

Complex objects are similar to container classes in that they are comprised of a collection of other objects sharing something in common via multiple inheritance. For example an airport might be represented as a complex object in a 2D space as a point, a line and a polygon representing a beacon, runway and a hangar, respectively (figure 6).

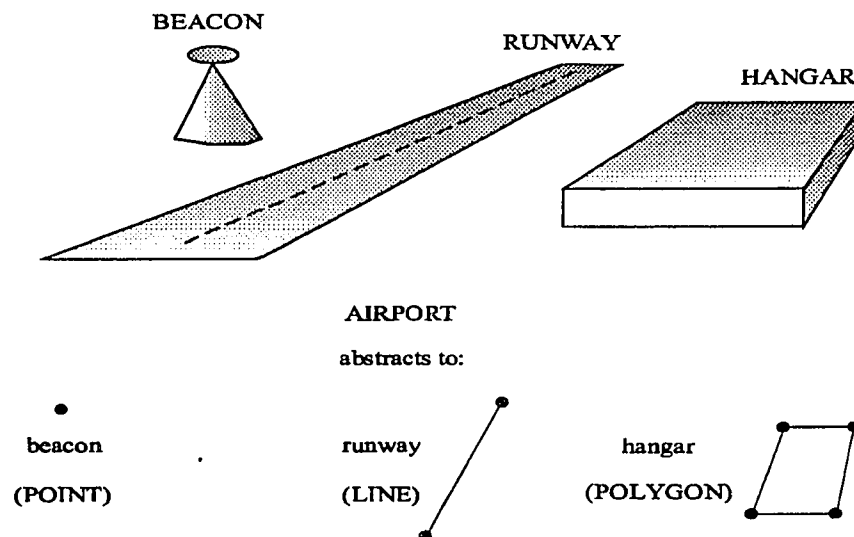
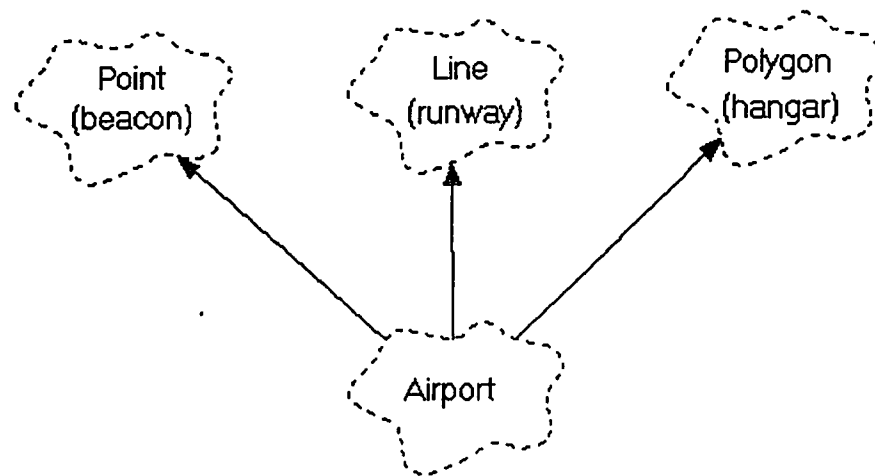


Figure 6. Complex objects.

The complex Airport class is derived from instances of three other classes: Point, Line and Polygon (figure 7).



**Figure 7.** Class diagram of a complex object.

## CHAPTER III

### PARALLELISM

#### PARALLEL ARCHITECTURES

There are currently two predominant architectures in use for parallel computing. These are single instruction, multiple data-stream (SIMD) and multiple instruction, multiple data-stream (MIMD). Most of the available hardware falls into one of these two categories. Each has its relative advantages and disadvantages with respect to programming and the types of problems to which it is best applied. These systems are referred to as fine-grained or coarse-grained. This is determined by the number of processors available, and how much memory is allotted to each processor. Fine grained systems have more processors than do coarse grained systems. In general, coarse-grained systems have tens to hundreds of large processors, and fine-grained systems having thousands to millions of (usually) small processors.

A fine grained system, just because it has more processors, is not necessarily faster than a coarse-grained system. Although more processors indicate a higher degree of parallelism, if they are small processors (limited memory), the additional communications overhead may offset the gain in computational power. In many cases, a single large processor may be faster than many smaller ones. Also, some algorithms are inherently sequential and do not lend themselves as well to parallelism and will work better on a single large processor. This is why most fine-grained parallel machines are designed to work under a single-processor host.

### SIMD

The SIMD architecture requires that a single program instruction is given to each of the available processors along with a portion of the data. Each processor then operates on its allotted data independently. SIMD parallelism is sometimes referred to as "data-parallelism". All the processors are concurrently performing the *same* instruction (figure 8). When all of the processors have finished their respective operations, they can all be reassigned a new operation. Processors finishing their respective operations before others are done must "wait out" until the others finish and all the processors can accept a new instruction. The disadvantage here is in the wasted computational power due to idle time. This architecture has been the most commercially viable to date due to the relative ease with which SIMD machines can be programmed for.

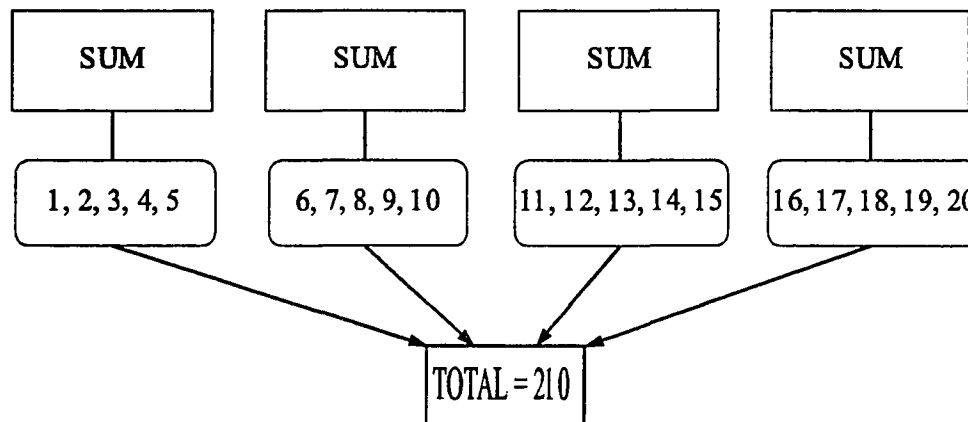


Figure 8. SIMD architecture.

### MIMD

Unlike SIMD, MIMD processors can perform dissimilar independent operations on their allotted data. This means that different processors are concurrently performing *different* instructions on either similar or dissimilar data (figure 9). This architecture is sometimes referred to as "task-parallel". MIMD systems can differ in the way they access



memory. The two types of memory access methods are shared memory and distributed memory. In shared memory multiple processors look into the same area of memory for data. With distributed memory, each processor is allocated a portion of memory for its own exclusive use. MIMD computers are often referred to as cubes or hypercubes, due to their three dimensional physical memory arrays. MIMD computers are *array* processors, whereas SIMD machines are *vector* processors. MIMD computers can be used in SIMD configuration, but not vice-versa.<sup>1</sup>

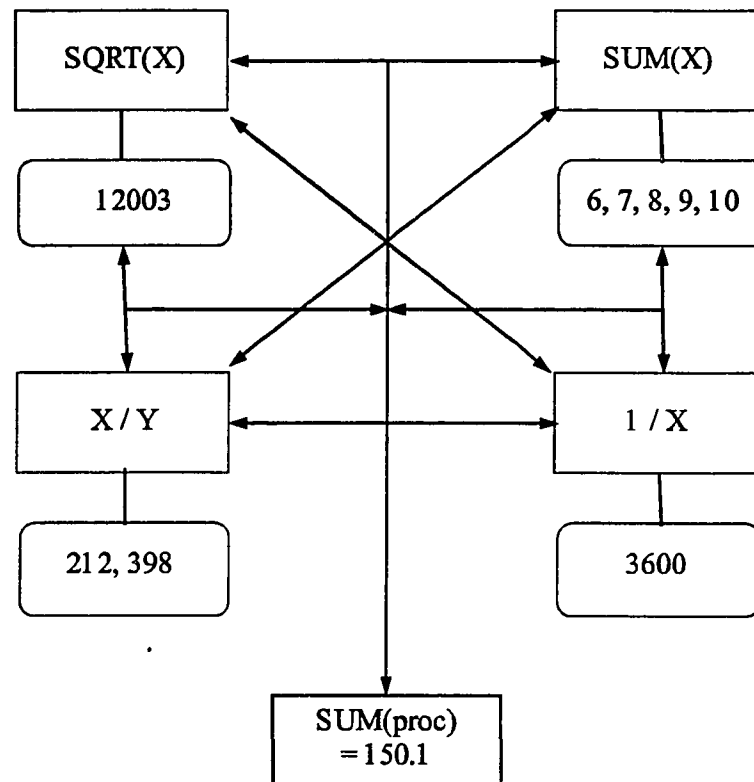


Figure 9. MIMD architecture.

Additional complexity occurs in distributed memory MIMD systems when the

<sup>1</sup> Vectors of SIMD computers can be added together, resulting in a processor array which would then be a MIMD configuration.

memory distribution topology is reconfigurable as opposed to fixed. This means that different portions of memory are allocated to different processors at different times during program execution. This stochastic topology becomes a problem in that considerable overhead is required to track memory allocation and availability. Inter-processor communications become more difficult to route when the destinations are "moving targets". There is substantial research being conducted currently in the area of memory reconfiguration. A fixed memory configuration is easier to program for in that a given processor's memory is always of a given size at a given place, but is not as efficient in utilizing computing potential. A topology which can be configured to match the structure of an application will be the most efficient in executing the application.

The communications necessary between processing nodes can be cumbersome. If node  $e$  requires a product from node  $q$ 's operation, node  $e$  must not begin its operation until it has received that data. Some method for controlling data flows between nodes must be considered. Algorithmic efficiency is a major concern with respect to inter-processor communications. The trade-off between performance gains due to adding processors and the performance losses due to communications between nodes must be considered when either designing new parallel algorithms or adapting procedural ones to the concurrent realm.

## DECOMPOSITION

Decomposition refers to breaking down a problem to concurrent algorithms and, ultimately, the mapping of a problem to the processor nodes. The decomposition method and the resulting algorithm are highly problem-dependent. Requirements for inter-processor communications are dependent on the method used for decomposition. An object-oriented data model already has a head start on the decomposition process in that some decomposition has already been performed during the class construction process. There are three fundamental decomposition methods: perfectly parallel, domain and control decompositions.

### Perfectly Parallel Decomposition

Perfectly parallel decompositions require the least communication between processors. This is because the instructions and data are distributed to each processor, which performs its assigned operation autonomously and broadcasts the result at the end of its operation. This differs from SIMD in that with a SIMD configuration all processors perform the same instruction in unison as opposed to the MIMD configuration distributing different instructions to the nodes. An example of perfectly parallel decomposition would be to assign each node of a graph to an individual processor. Each processor then is assigned a function. For some processors the function might be FindNorthNeighbor, for others FindWestNeighbor. This type of decomposition is simple and effective, but is best suited to coarse-grained systems where each processor can handle a large and complicated instruction (program).

### Domain Decomposition

This type of decomposition is appropriate for large, static data sets such as large matrices. The matrix can be broken into sections, calculations can be performed in parallel on each section, then the local section solutions can be combined into the global solution.

Finite element analysis is a good example of domain decomposition. The entire domain (structure) is broken into subdomains. Calculations are performed on the individual subdomains and then are combined together to form a solution for the original domain. Domain decomposition can also be applied to problems where the data is dynamic, but is tied to a single entity. An example of this type would be dynamic allocation of candidate supply nodes over a fixed network in a location-allocation problem.

### Control Decomposition

Control decomposition is applied when the problem has no obvious "breakpoints" in the data. Conversely, the breakpoints occur in the procedure(s) being used on the data. The two main flavors of control decomposition are functional control decomposition and manager/worker control decomposition.

Functional decomposition is performed by taking a problem apart according to the various tasks associated with solving the problem as a whole. For example, a procedural program is composed of algorithms which perform a specific task on some part of the data. An example of functional decomposition would be taking apart the steps required to perform an overlay analysis on some spatial object. For example, one might first wish to buffer several objects then overlay the buffers to find the intersection area. This process would be decomposed into the buffering process, the overlay process and the intersection process.

Manager/worker control decomposition is a form of functional decomposition in

that a manager process controls many subprocesses. Operations are performed in parallel on the subprocesses. Results of these operations are then passed back to the manager process which then processes the results, sends its result on and then reapports subprocesses once again. This is analogous to the main body of a sequential program calling subroutines.

### Layered Decomposition

Layered decomposition is considered to be a large scale decomposition method. Large scale decomposition is required when one of the previously outlined methods is not singularly adequate. The first step is to decompose the software to be developed into modules, then use an appropriate decomposition method on each module. This is analogous to large scale functional decomposition.

The first layer (corresponding to a module) might employ perfectly parallel decomposition. The results of that process would then be passed down to the next layer where domain decomposition is employed. The various decomposition methods can be used as appropriate to a given layer.

### Parallelism for Neural Networks

The term "massively parallel" shows up in most of the literature associated with neural networks. Does this mean that massively parallel architectures are inherently neural? Can neural networks be implemented on any parallel computer? What does "massively parallel mean"? These issues will be dealt with in greater detail in chapter 5, but for now, the answer to these questions is: it depends on the number and connectivity of the processors.

A completely connected architecture will most certainly be able to accommodate neural network applications (or any other application). A completely connected

architecture requires that every processing node be connected to every other processing node in the processor array, and (with path redundancy) is the most robust of connection topologies. It is the current belief that this topology most closely models the neuron/synapse arrangement of the human brain.

Other topologies can be configured to best suit the individual neural model being applied. As we will see in chapter 5, there are really only a few models currently used in neural network applications, and most can be implemented sequentially on a single processor (although such an implementation defeats the advantages gained by the neural approach due to its inherent parallelism). Architectures employing four or more nodes are considered to be massively parallel. Techniques used on four processors can be scaled to work on four thousand processors, or more. Both SIMD and MIMD architectures are massively parallel and are being successfully used for neural network applications.

#### The Appropriate Parallel Architecture for GIS

Verts [Ver88] and Healey [Hea89] have studied and reviewed the various architectures currently available from which GIS applications could be developed. Healey suggests that the MIMD architecture utilizing reconfigurable, distributed memory is best suited to GIS. This configuration allows program instructions to be mapped to individual processors, each with its own private memory, with the topology of the network definable on-the-fly. Given a solution to the stochastic memory topology problem, this may be the architecture of choice.

The SIMD architecture, while simpler to program for, does not promise throughput potential which could be ultimately attained from a properly configured MIMD environment. Additionally, it may not ultimately lend itself as well to true object oriented parallel programming for spatial analysis. It is interesting to note, however, that

the only commercially available neural coprocessor is SIMD-based.<sup>1</sup> This is probably due to the simplicity of programming for the SIMD configuration and that the two most common neural network algorithms (backpropagation and Hopfield networks) only require SIMD architecture. As the field of neural networks evolves, MIMD architectures may become more visible in both the literature and the vendor community.

Before the question of appropriate architecture should be answered, the proper question to ask is: how does the functionality of GIS and spatial analysis decompose for parallelism? The holistic view will lead us to layered decomposition, for sure. But which style of decomposition can be most often applied, and what are the communication/performance tradeoffs associated with the related topology? What if we are not able to reconfigure the topology configuration "on the fly", and must select a single, hard-wired topology for reasons of cost?

In contradiction to Healey and Verts, Li [Li 92] suggests that data-parallelism is optimal for GIS due to the natural ability to model data with regular spatial and temporal structures (as modeled in raster GIS'). Li [Li 92] offers examples of a very successful multiple-SIMD implementation in C\* ("C star") running on a CM2 (second generation connection machine) of fundamental raster-based GIS operations. In this light, a "90/10" approach seems reasonable. That is, if some SIMD configuration  $\alpha$  meets the connectivity needs of 90% of the decompositions and the other different needs amount to only 10% of the decomposition techniques, then topology  $\alpha$  should be the topology of choice.

Certainly a single, hard-wired SIMD topology will be cheaper and easier to program for. Due to these factors, it is probable that the first commercially available parallel GIS will be SIMD-based. If this system can get into the hands of the most users in the shortest amount of time and is maybe only 30% less efficient than would be the MIMD reconfigurable topology system, an argument could be made that this would be the

---

<sup>1</sup> Adaptive Solutions, Inc., Beaverton, Oregon.

appropriate architecture for GIS. So, the answer to the question of which is the best architecture for GIS may be philosophical rather than technical. Never-the-less, this dissertation will attempt to forward one approach, and the sages of the industry are free to argue its correctness.

### Object - Parallelism

As mentioned in the previous chapter, object-oriented data models have already been decomposed once during the class construction process. Class decomposition can be considered a formalization of domain decomposition, control decomposition or both [Rag91]. The goal of class construction and of decomposition is to break a problem up into appropriate objects. Therefore, taking an object-oriented approach to application development is the first step in the process of parallelization of the application.

### Parallel Objects

As we saw in chapter 2, objects are abstractions that tie together data and procedures for operating on that data. Proper object oriented class construction can and should leave only the object-to-processor mapping to be assigned (remember from chapter 3 that decomposition breaks up a problem and mapping assigns parts of the problem to specific individual processors). The mapping should be very straightforward and simple if the decomposition is appropriate for the problem.

It is quite likely for objects to be mapped directly to individual processors (as in the example in chapter 3 of assigning graph nodes to processing nodes). This is why the distinction between decomposition and mapping must be made clear. While good decomposition leads to simple object mapping, the decomposition step does not explicitly involve mapping.

Good parallel objects should be decomposed with the architecture's granularity in



mind. For example, if the target system is coarse grained, the procedural objects might be more complex than would be those for a fine-grained system. Similarly, data objects for a fine-grained system should contain less data than would be permissible for a coarse-grained system. Such considerations will make for better decomposition and efficient mapping. The most fundamental objects with respect to processor mapping are arrays: 1-D (vectors) for SIMD, 2-D (arrays) for MIMD and 3-D arrays for MIMD hypercube topologies.

The relationships between objects should be reflected by the connection topology between processing nodes. For example, in a MIMD configuration, if an object has a "DrawToScreen" function, and there are many instances of the object distributed over many processors, each processor must have a connection to the processors which are performing tasks related to *DrawToScreen()*. Some nodes may be performing perspective transformations on 3-D points which are the objects to be drawn to the screen. Other nodes may be concurrently differentiating the points over some function in time. The topology must allow the results of these computations to be communicated between objects (nodes). Therefore, objects which are concurrently distributed over the architecture are considered to be parallel objects.

## CHAPTER IV

### TOWARD AN OBJECT-ORIENTED DATA MODEL

#### OVERVIEW

The following sections trace the development of object-oriented spatial data models through the recent literature to date. The different models are initialized for ease of referral in the succeeding sections. The descriptions of the models are simplified or abridged wherever possible in the interest of brevity and clarity of this dissertation. The models are divided into three groups: basic object-oriented, knowledge-based and spatio-temporal.

#### Basic Object-Oriented Models

These models represent the first step from the current generation of entity-relationship based models toward a true, ground-up object-oriented spatial data model. The approaches presented in this section are primarily concerned with the basic shift toward objects as opposed to the complete spatio-temporal models presented later in the chapter, which are more highly developed and robust models.

The G model. As mentioned in chapter 2, several attempts have been made to develop a separate object-oriented spatial data model and then combine that model with a traditional RDBMS. Gahegan et.al. [Gah88] devised a sort of black box approach where an "object system" operates in conjunction with a more traditional "spatial system". In their model an object analyzer handles class hierarchies and data retrieval methods. A

rule-based inference engine then directs data and/or queries to standard structures such as quadrees, b-trees and I/O.

Because the object processor is not integral to the underlying data structure, it would seem that an unnecessary logistical overhead has been imposed in order to aid abstraction. Instead of directly operating on data, queries must first pass through the middleman (the object processor) in order to be reassigned to the proper data or algorithms. It would seem that late binding provides a way around this by effectively spreading out the object processor through the code at compile time, then making the appropriate query/data assignments at run time, thereby taking full advantage of encapsulation via container classes. Gahegan et.al. [Gah88] could not make use of late binding in their model because the target system was written in C in conjunction with relational INGRES.

The IFO Model. Worboys et.al. [Wor90] suggested a hybrid object-relational model for spatial databases based on Is-A relationships, Functional relationships and complex Objects (IFO) originally introduced by Abiteboul and Hull in 1984. This approach is strictly logical in that no attempt is made to map the logical structure to a physical model found in any currently available DBMS or any object-oriented programming language.<sup>1</sup>

The IFO model begins with atomic objects (simple objects) of three types; printable, abstract and free. Printables are integers, reals, strings and pixels. Abstract atoms are non-printable conceptual objects. Free atoms are links between objects showing generalization and specialization. Next, atomic objects are related through the hierarchy

---

<sup>1</sup> A tomato Is-a vegetable. It's Functional relationship is to serve as nutrient for humans.

by is-a relationships as found in the Codd Entity-Relationship model. Objects are also related according to function or behavior. For example, polygon objects are topologically related in that they all are constructed of chains having left- and right-polygons. Throw this all together and you get a complex object.

If this nomenclature isn't confusing enough, the authors introduce a symbology for diagramming structures using circles, boxes, diamonds and arrows which is context-sensitive. While this model attempts to ease abstraction, the analyst would be bogged down by having to keep track of the current morphic state of an object of interest at any or all levels in the hierarchy.

It is worthy to note, however, that the approach taken by Prime-Computervision (*sans* confusing symbology) for their System 9 GIS spatial data model is very similar in underlying concept to the IFO model. This is especially true with respect to shared topology among spatial objects. System 9 has mated a hierarchy of spatial objects with the relational DBMS Empress. As a result, System 9 is the best GIS currently on the market for handling many inter-related complex objects without the excessive storage redundancy typical of layered systems [Rex91].

### Knowledge-based Models

Knowledge-based systems are founded on the concept that there is unique information or relationships between entities that is critical to the analysis of a set of related entities. The relationships are explicitly modeled and analysis is performed by tracing relationships through some topology constructed from the taxonomy of the entities and their relationships.

The Z-M Model. Zhan and Mark [Zha92] propose and implement a model based on categorizations and representations of spatial knowledge through the

specification of core classes and their associated relationships. This approach is a variant of the semantic network model. Specifically, the model is concerned with representing spatial knowledge as it is comprised of geometric-topological knowledge, geographic phenomena and algorithmic-symbolic knowledge (algorithms and heuristics applied to the spatial data). The core spatial classes put forward are the geometric primitives: point, line and polygon. The "visual variables" at the top level of figure 10 represent the knowledge related to geographic phenomena. The algorithmic knowledge is contained within the core classes as methods. Figures 10 and 11 below diagram the model as presented by the authors.

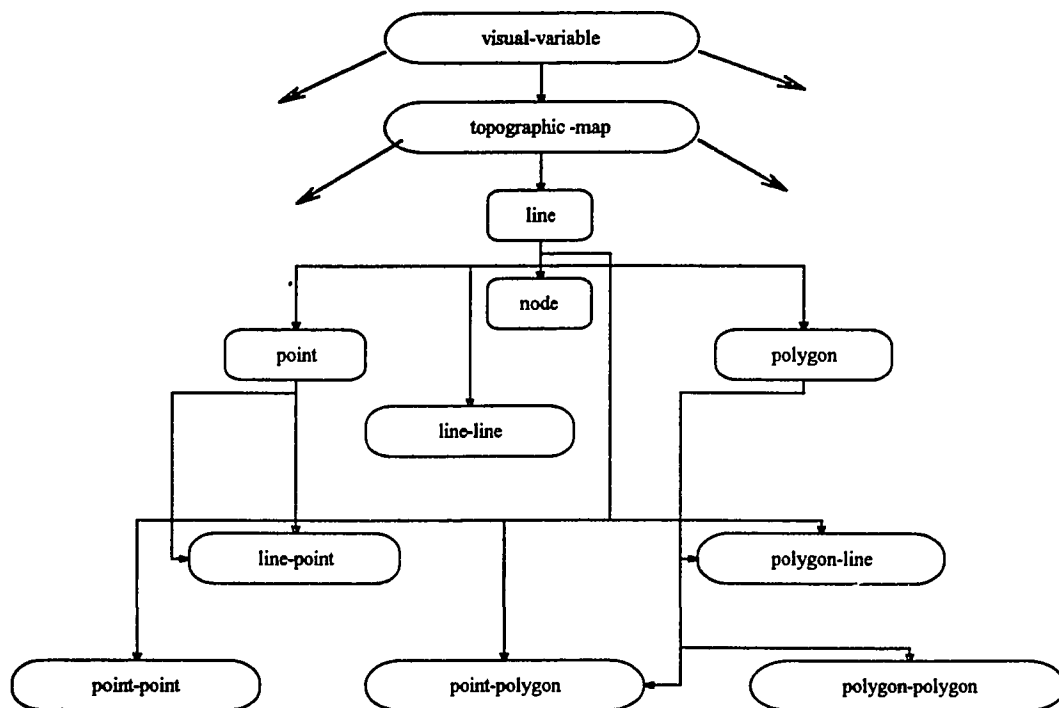
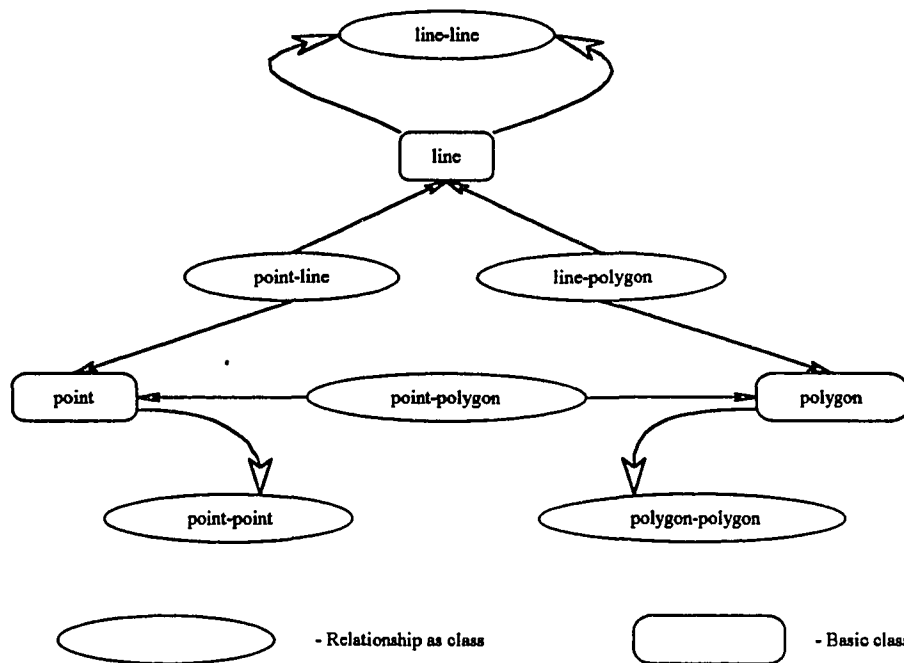


Figure 10. The Z-M class hierarchy.

The arrows pointing off into space represent connectors for new, user-defined classes. Although the authors did not take the diagram to its full extent, the model sets the stage for the proper objected-oriented representation of complex objects. For example, the next class layer in figure 10 would be "point-line-polygon", which would correspond to the abstracted airport in Chapter 2.



**Figure 11.** The Z-M model relationship hierarchy.

Most notable about this model is that it is 2-dimensional and that the fundamental base class is the Line class (as opposed to the Point class). Additionally, the use of the "Is-A" relation in the hierarchy is one-way as in the IFO model and is therefore not as robust as the E-F model (e.g. "point" is-a "line" because line is the base class). The graphical notation needs to be further developed and is not very intuitive in its current state. The authors give an example of the model implemented in a C++ - derived language "CLIPS/COOL". The implementation is concerned with exercising the geometric-

topological aspect in test DLG files for geometric and topologic consistency, and seems to be successful in that application.

In conclusion, the Z-M model is more oriented toward a specific artificial intelligence (AI) implementation in support of geometric-topologic knowledge and does not yet specifically concern itself with either multi-dimensionality or the temporal component, as acknowledged by the authors who offer the model as a platform for further research. The semantic network model, while offering promise in the analytic area, does not yet appear to be the appropriate choice for a robust system design.

The M-R-B Model. Mackay, Robinson and Band [Mac92] propose an interesting twist to the knowledge-based approach through an explicit tie to graph theory (as opposed to the implicit relationship of semantic networks). In the M-R-B model, a query system is derived from the graph of the relationships between entities. In the example presented, they abstract a 3-D topography to a collection of topographic objects, which can first be represented by a tree. The tree is then represented in graph form, and the simulation and query system model is based on the graph. An example of this abstraction is given in Figures 12 and 13, below.

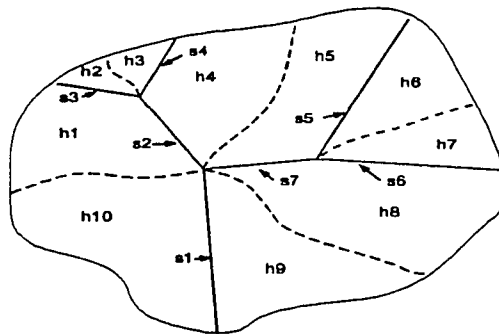
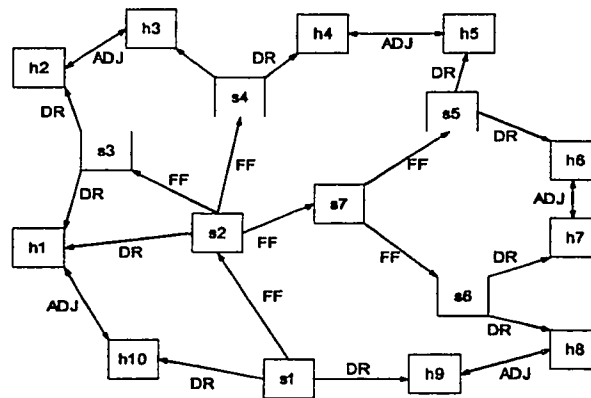


Figure 12. A drainage basin.



**Figure 13.** The M-R-B model representation of a drainage basin.

The entities labeled *h* represent hillsides and the entities labeled *s* represent streams. Figure 11 shows the topography of the sample data, where the streams drain the hillsides. The dotted lines correspond to ridges between hillsides. Figure 13 depicts the abstraction of the topography and the relationships between hillsides and streams to a graph. The nodes represent the topographic objects and the edges show the hierarchy of stream drainage (FF = "flows from"), the topology of the streams and hillsides (DR = "drains"), and the functional relationships between hillsides and streams (ADJ = "is adjacent to").

Queries can then be made by simply selecting objects. Their complex relationships (as modeled) are the primary attributes associated with the objects. The model relies on the property of transitive closure along graph paths for query correctness. This means if stream *s1* flows from stream *s2*, and stream *s2* drains hillside *h4*, then stream *s1* [indirectly] drains hillside *h4*. This is analogous to a class hierarchy where dependent class inherit attributes from base classes via intermediate base/dependent classes.

Some of the pros and cons of this approach are that complex topologic *objects* are more difficult to model, but complex *knowledge* is more easily represented than by



standard relational databases. As a result, complex *queries* are optimized. Again, it seems as if this approach applied on top of a more robust topologic- and geometric-primitive object schema might provide the best of both worlds.

### Spatio-Temporal Models

Spatio-temporal models are distinguished from the other models presented by their focus on the inclusion of the third Euclidean dimension and a fourth dimension, time, in the model. Ingredients from these models comprise the most likely basis from which future spatial analysis systems will evolve.

The method of dealing with time in the current generation of 2.5-D relational GIS' is to represent entities effected by time in separate layers, referred to as snapshots. The snapshot approach is both inefficient and cumbersome. Additionally, snapshots fail to address the continuous nature of time.

Hazelton [Haz92] presents several metaphors for the temporal component in GIS. Most notably, he compounds the problem of representing the continuity of time by pointing out that time is probably fractal by nature. That is, time is not linear, but changes over different intervals. Although this adds conceptual complexity to dealing with time, it may simplify the modeling of time if eras can be formulated.

The E-F Model. Egenhofer and Frank [Ege90] have been evolving a model since 1984. The instigation for their work stemmed out of the lack of available systems to effectively handle complex spatial objects at different levels of abstraction. They have suggested an objected-oriented model emphasizing inheritance and propagation which is more powerful than the standard relational model due to generalization and aggregation properties of inheritance (chapter 2).

The authors refer to propagation as a form of multiple inheritance which minimizes redundant storage of data shared between objects sharing base and derived classes. This is another form of vertical integration of spatial data [Rex91] where calculated data for a derived object is aggregated from related data in the base class(es). For example, the population for an instance of a county would be calculated "on the fly" by aggregating the populations of the individual towns contained in the associated town base class, rather than being stored as an individual data object.

The authors formalize propagation as a general method implemented via the system-specific operational language (a Standard Query Language-like predicate calculus). The example offered is reproduced below:

**\*p (PartOf, SettlementPopulation, CountyPopulation, BySumming).**

Parsing this line would return in English: "propagate the county population by summing the settlement populations which are contained within the county".

The E-F model suggests that an object-oriented GIS be composed from at least four fundamental base classes. These base classes are: 1) the db-persistent class, 2) the spatial class, 3) the graphical class, and 4) the temporal class. The db-persistent class should contain objects related to database operations and data management. The spatial class should contain objects based on geometry and would include topological relationships between objects. The graphical class should contain objects related to display to screen, plotter and file output. The temporal class should contain objects pertaining to the history of the objects contained in the spatial and db-persistent classes. For instance, the temporal class might contain a method which tracks differences in topology over time, such as a voting precinct boundary.

The E-F model relies heavily on multiple inheritance to combine objects from the base classes into derived classes which more appropriately model real-world data and situations than do the current generation of relationally-based data models. The authors conclude their presentation of the model with a succinct summary of the primary distinction between the E-F model and the IFO model with respect to the "Is-A" relationship:

Three important conceptual differences exist between inheritance and propagation: (1) inheritance is defined in generalization (is-a) hierarchies, while propagation acts in aggregation (part-of) or association (member-of) hierarchies. (2) Inheritance describes properties [attributes] and operations [methods], while propagation derives values of properties. (3) Inheritance is a top-down approach, inheriting from the more general to the more detailed class. Propagation, on the other hand, acts bottom up.

The third distinction is the essence of the E-F model. Where the IFO model is a one-way inheritance hierarchy, the E-F model specifies a two-way data path. Specifically, in the interest of reducing storage, as much data as possible is derived from base classes rather than being represented explicitly at the level of the object of interest (as required by the IFO model). Although the model was not implemented at the time of publication, a straightforward implementation would be possible in a language such as C++, Smalltalk or LISP. Further work by the authors in the area of a meta-language supporting the E-F model would indeed be a worthwhile contribution to the literature.

The L Paradigm. Although Langran [Lan92] doesn't offer a complete spatio-temporal data model, a paradigm of temporality is presented which needs to be examined here. Langran suggests that three major temporal data types should exist in a spatial data model: states, events and evidence. These data types act as attributes of spatial objects which contain essential temporal information about the object.

State can be considered a basic object with events and evidence as its temporal attributes. State contains the geometry of an object at a given point (or period) in time. An object's geometry can change over time as a result of events. An event is an occurrence to or an action on the object at some time. Evidence is information about the event which qualifies the state of the object.

For example, a forest begins life in a GIS at state  $t_0$ , its event at  $t_0$  is "old growth". The evidence for this is by survey and aerial photography (Figure 14). Similarly, a road running along the forest has initial state  $t_0$  sharing evidence with the forest of survey and aerial photography.

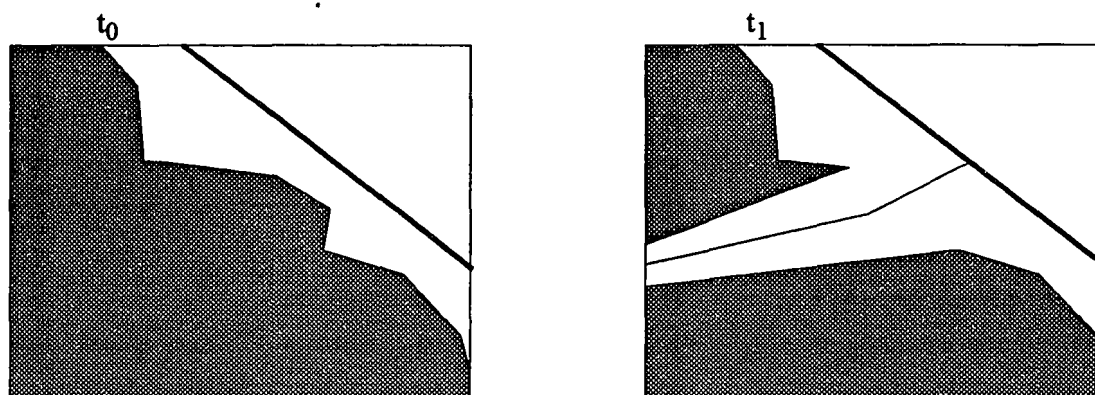


Figure 14. A road and forest at time  $t_0$  (a) and at time  $t_1$ (b).

At time  $t_1$ , events have occurred to the forest and to the road (figure 14b). These events are the cutting of the forest, the addition of a new road and the intersection of the new road with the previously existing road. The evidence of these events are the permits for the cutting and the road building, surveys, and aerial photography. By these means, the origin and pedigree of the new data is preserved. In a relational model the state would be a primary key, with the event being a secondary key and the evidence would belong to the event table.

This approach differs from the simple snapshot method in that the data model would have explicit temporal relations due to the inclusion of the three data types: states, events and evidence. Such an organization of data should help to reduce the redundancy of spatial data which results from maintaining snapshot layers. This could be accomplished by linking the state geometry to event periods. The geometry from one state could be combined with the geometry from a successive state so that the common geometry needs to be stored only once (figure 15).

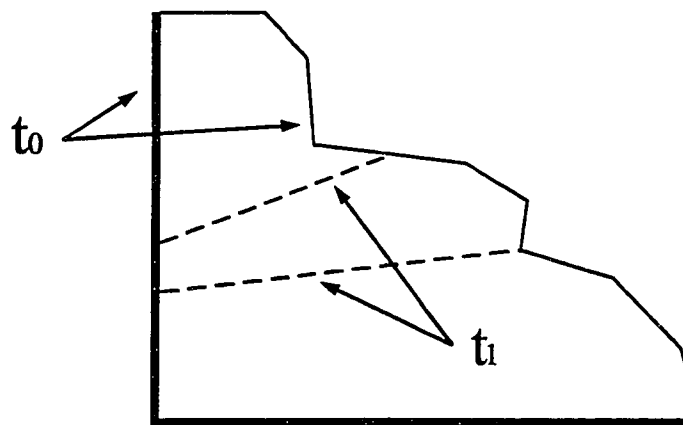


Figure 15. Combined geometry of two time states.

In figure 15 the geometry for the forest entity at state  $t_0$  is initially stored (solid lines). At time  $t_1$  only the geometry for the event (change) is stored (dashed lines). A relationship is established in the data model so that when state  $t_1$  is to be displayed, the geometry is built from what is stored for state  $t_1$  and the geometry that is linked to  $t_1$  from other states, in this case, portions of the geometry for state  $t_0$ .

The Pr Model. Price [Pri89] presents a semi-object-oriented spatio-temporal model for land parcels giving more explicit attention to the temporal component than does the E-F model. Although the model is yet another variant of the snapshot approach linked

to a relational database, it (like E-F) utilizes the properties of inheritance to model temporal relationships between base class and derived class objects.

For a cadastre, time is handled as an object attribute via an abstract data type, "date" which is a data element of an instance of a "lot" object's data structure. The "transaction" data type acts as an object of a class containing information concerning transaction on lots. Transactions have a date attribute as well as lot attributes. This approach is very similar to the L paradigm's "states" and "events".

The model only supports two operations: updates and queries. When lots are subdivided the new lot is derived from its base lot. The database is updated with the information pertaining to the creation of the new (derived) lot such as its ID, geometry, ownership and date of subdivision. Pointers back to the base lot are included in the record for the new lot. A pointer to the new lot is appended to the base lot record. Similar operations are performed when transactions occur.

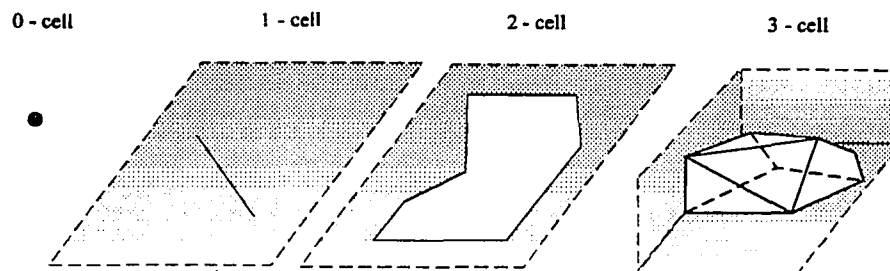
Temporal queries can be made via the relational database by specifying the data attribute as a primary key. For transactions, the transaction type might be the secondary key. By using pointers to and from the "lot" and "transaction" objects, the historical information associated with each data type is embedded within the database due to date being a part of the data structure (an attribute of the "lot" and "transaction" objects).

The Pi Model. Pigot [Pig92] first rigorously presents a 3-D topological model, concentrating primarily on the topological relationships in the geometry of spatial data. He then extends the model (with Hazelton) to 4-D with the inclusion of the temporal dimension through snapshots. The continuous nature of time is addressed by "integrating" the snapshots. This model is especially significant to this dissertation

because Pigot was the first to present a general model to the GIS literature from a purely computational geometry perspective.<sup>2</sup>

The model's fundamental objects are "manifolds" and "cells" in 3-D Euclidean space. Their topological relationships in 2-D+ can be modeled with graphs, resulting in a "cellgraph" model (as in the M-R-B model).<sup>3</sup> A "manifold" corresponds to an  $n$ -space. A subset of the "manifold" is a "cell". Cells are the true primitives of the model, whereas the manifolds exist to define the feasible space in which cells may exist.

A cell is 0, 1, 2 or 3 depending on its dimension in  $R^3$ . An 0-cell corresponds to a point, a 1-cell corresponds to a line, a 2-cell is a polygon and a 3-cell is a polyhedron (figure 16). Cells of different dimension can be combined across manifolds to form "complexes".



**Figure 16.** Cell complexes.

<sup>2</sup> There have been earlier models and applicable data structures presented in the computational geometry literature, but, until recently, that literature really hasn't been standard reading for GIS professionals.

<sup>3</sup> 0-dimensional objects (points) simply exist, unconnected, in  $n$ -space.

The topological properties for these cells are comprised of the familiar Boolean algebraic set operators such as union, intersection and containment as well as the spatial relations proximity and adjacency. Cells can be organized according to the rules for these operators to form complexes in  $n$ -space. For example, a complex in a 3-manifold results from the union of a 1-cell and a 2-cell if the 1- and 2- cells originate from different manifolds. Although the proofs for this topology are rather complicated, the general concept in  $R^3$  is quite straightforward and intuitive.

The model becomes a bit more difficult conceptually (as do the proofs!) when the fourth dimension is added. However, the operation set valid in  $R^3$  works in  $R^4$  as well when treated as an additional dimension in Euclidean space. The fourth dimension can be applied to all types of cells, given a temporal delta for an  $n$ -cell. Figure 17 attempts to illustrate this concept for the 0- and 1-cell primitives. The illustration is extensible to 2- and 3-cells (omitted here because the representations in 2-D may be confusing).

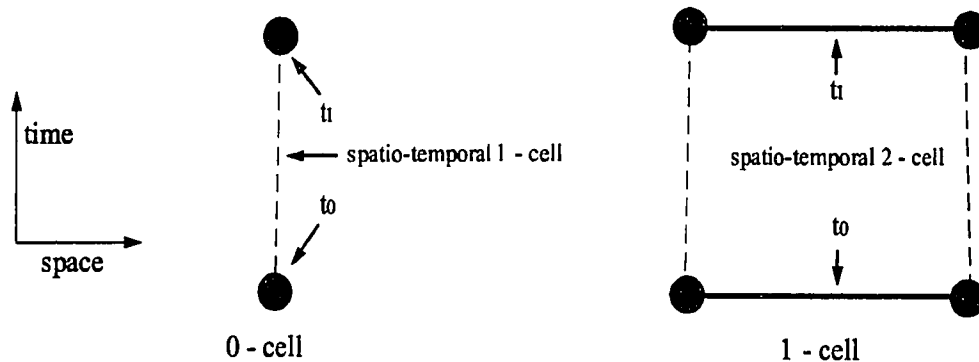


Figure 17. Spatio-temporal cell complexes.

The "snapshot" is defined by the geometry of an  $n$ -cell (or topology of an  $n$ -complex) at time  $t_1$ , and the topology of the spatio-temporal  $n+1$ - cell (complex) is



defined by the difference  $t_1 - t_0$ . In this context, time is treated as a continuous entity,  $t$ , with discrete times  $t_1$  and  $t_0$  acting as boundaries to the time space (the  $n+1$ - manifold).

The advantage of this approach is that standard 2-D spatial analytic techniques can be applied in 4-D without the need for a new metric. While this model presents a sound topological and geometric framework, no suggestions are put forth with respect to optimal data structures required for implementation and attribute associations, not to mention a comprehensive object class taxonomy.

The W Model. Worboys [Wor92] has proposed a purely object-oriented spatio-temporal model. The underlying assumptions for this model are that the geometry and topology are embedded in 2-dimensional Euclidean space and that the temporal dimension is handled as attributes to spatial objects. In this model a spatial object occupies space in 2-D depending on its temporal attribute(s). The unary temporal dimension is considered to be orthogonal to the Euclidean plane containing the geometry. It is additionally assumed that temporal changes occur at finite, discrete points in time, and that time is continuous and linear. The 2-D space is "projected" through the unary temporal dimension for analysis.

Although this model is not as topologically robust as the Pi model, it is presented in object-oriented terms and makes a contribution to the model developed in chapter 8. The basic approach taken is to "project" geographic objects onto spatial and temporal objects to form spatio-temporal or "ST" objects. An algebra is then employed for the analysis of "ST" objects in  $R^2$ .<sup>4</sup> Although this model has not been developed to its potential, it is fundamentally intuitive and is a step in the direction of simplifying problem abstraction.

---

<sup>4</sup>  $R$  = set of real numbers.

Geometrically and topologically, the W model is similar to the Pi model in  $R^2$ . Instead of  $n$ -cells, simplexes are used. Simplexes are geometric primitives combined to form complexes. In  $R^2$ , the primitive simplexes are the 0-, 1- and 2-simplexes, which correspond to the familiar point, line and polygon geometry. A complex can be comprised of simplexes which differ temporally. The temporal is added, forming 0-T-, 1-T- and 2-T-simplexes. These can be identically represented by the  $n$ -cells depicted in figure 17. Figure 18 illustrates a complex comprised of simplexes having different temporal components. Note that the large triangular surface of the simplex is subdivided into two simplexes forming the complex due to the differences in the simplex's temporal component. These decomposed surfaces are referred to as temporal "atoms".

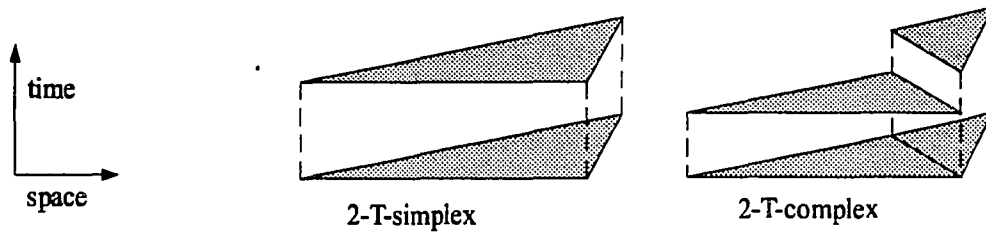


Figure 18. Spatio-temporal simplexes and complexes.

The hierarchical structure of ST objects is organized as shown in figure 19. The base class is the abstract "spatial" class. Its derived classes are the "point" class and the "ext" class. The "point" class contains simple x-y coordinate pairs and the "ext" class contains extended objects in 2-D comprised of point sets. It is of interest to note that the temporal component is not explicit in the class hierarchy.

The obvious shortcoming of this model is that it does not address geometry and topology in  $R^3$ . While it provides a sound foundation for extending the model to a true 4-D model, the work has yet to be published. As it stands however, it is a good 3-D (in  $R^2$

+ time) model given that the third dimension is temporal and exists in attribute space projected onto the third dimension of  $R^3$ .

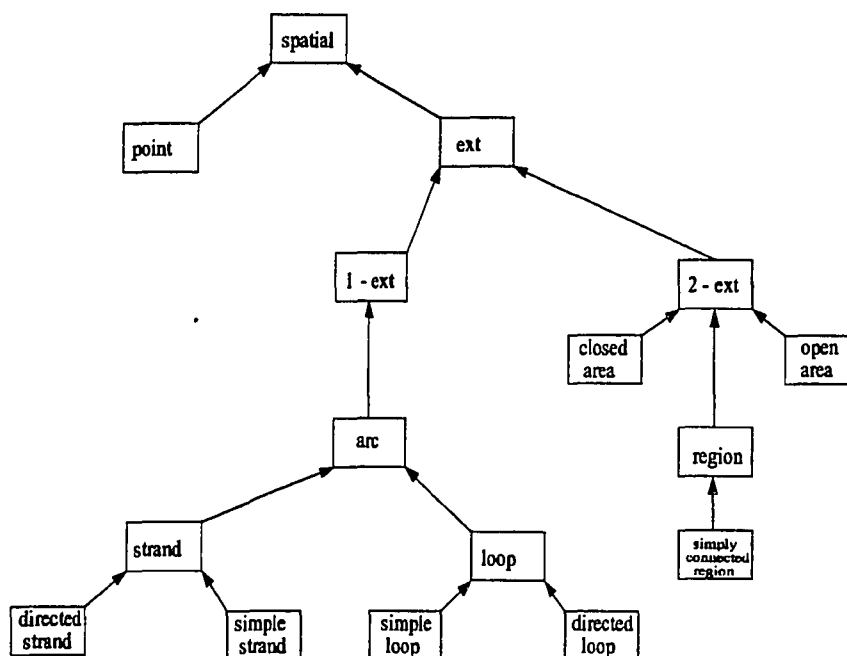


Figure 19. The hierarchy of ST objects.

## CHAPTER V

### THE OPSTAMS DATA MODEL

#### MODEL OVERVIEW

As we have seen from the previous chapter, several basic themes have emerged from the literature with respect to desired features and functionality of spatial analysis systems. They should be object-oriented to aid abstraction and queries. They should be able to represent knowledge. They should be at least four-dimensional and have the potential to handle continuous time as well as discrete time. The topology of the data model should be continuous over four dimensions. Additionally, they should take advantage of the recent advances in both hardware and software techniques, most notably parallel computers, object-oriented programming, and neural networks.

Models have been developed to address these issues one or two at a time, but the literature does not yet offer a comprehensive model which addresses these issues holistically and simultaneously. The Object-Parallel Spatio-Temporal Analysis and Modeling System (OPSTAMS) approach presented in the following sections will attempt to unify these issues under a single, comprehensive model. The underlying assumptions for the model are that there are no implementational constraints for the model with respect to data storage and that the host architecture is massively parallel.

#### Computational Geometry and $k$ -Dimensionality

The geometric framework for the OPSTAMS model comes from computational geometry.<sup>1</sup> The methods of computational geometry are  $k$ -dimensional, where  $k$  is defined

---

<sup>1</sup> Computational geometry is a fairly new discipline which is a combination of computer science and classical geometry. For more information see [Pre85].

by the problem space. This dissertation will only be concerned with  $k = 4$ , however, because the model is consistent with computational geometry, it is extensible to  $k = n$ , as needed. The four dimensions concentrated on here are the usual three Euclidean spatial dimensions and time, which is assumed to be linear.

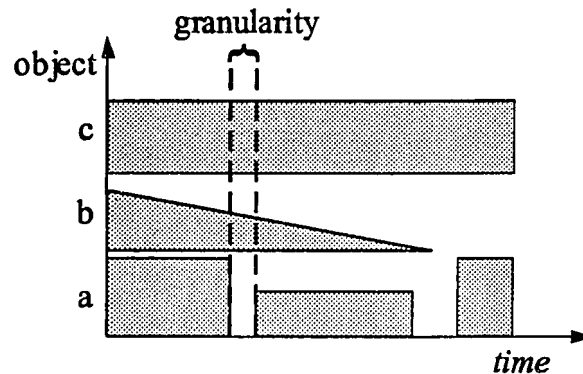
Computational geometry is primarily concerned with sets of points in space. Points are usually embedded in hyperplanes, halfplanes of which can belong to polytopes (convex polyhedra) of 4 (or more) dimensions. These are similar to the familiar points, lines and polygons encountered in 2.5-D GIS, but instead exist in  $k$  dimensions. The paradigm of computational geometry provides transitive closure of the topology of the geometry in 4+ dimensions [Pre85][Hoc61]. A spatial data model developed around the lemmas and theorems of computational geometry in 4 dimensions is easily extensible to as many dimensions as analysis requires.

### Visualization of Temporal Data in the OPSTAMS Environment

This model takes a different approach to the visualization of temporality than has been examined thus far, in that time is treated both statically (snapshots) and dynamically (animation). Time can be handled with a method that is a function for time, be it an integral or a formula for the fractal dimension. Every spatial object has a *time()* method associated with it. The inclusion of temporal methods as part of the primitive data structure assists in simulation, especially where interpolations of geometry are required as a function of time. The method may return a static value or function of time with respect to  $x$ ,  $y$  and  $z$ . The spatial objects for a given space are displayed accordingly.

A quantity of time with different starting and ending times is an era. Eras are either returned by the *time()* method of an object (or objects), or by the user. The user may specify how time is handled during a session. If time is statically set to some arbitrary point in time such as March 20, 1974, all spatial data extant for that time is displayed (at

the day-level of temporal granularity). Because time is treated as a member function to a spatial object, it may be returned as a constant or varying over an interval as shown in Figure 20.



**Figure 20.** Granularity of time.

For display, an era may be specified by setting a beginning time and an ending time. The temporal granularity factor (step size) for the given space is set to the smallest temporal interval contained in any of the objects active in the given space. Visualization of an era is accomplished by an animation which displays all spatial data for each time  $t$  for the given space, where  $t$  is the granularity factor transformed to display time.<sup>2</sup>

#### Selection of Four-Dimensional Data from a Two-Dimensional Display

At this point the astute reader may be wondering how it is possible to interactively pick spatial data from the screen when the screen is constrained to 2-D, but the data exists in 4-D. This is accomplished by first defining the epoch (a point or era in time) and by projecting the 3-D data onto 2-D from two different perspective points. Each perspective

---

<sup>2</sup> For example, a five-year era with one month granularity may be animated in five seconds of display time. Display time is user-defined.

view is mapped to its own window. Each window holds one of the three spatial dimensions constant while the other spatial dimensions and time may vary. This technique works equally well for both raster and vector objects in  $R^4$ . Time varies due to the animation of objects in each of the display windows.

### Class Structures

The classes comprising the OPSTAMS model are developed from a few primary classes which represent the fundamental concepts of GIS and spatial analysis. The base classes are abstract classes in that they do not directly contain any explicit data or methods. These base classes are the GEOMETRY class, the DATA MANAGEMENT class, the ANALYSIS class and the VISUALIZATION class (figure 21). The immediately-derived

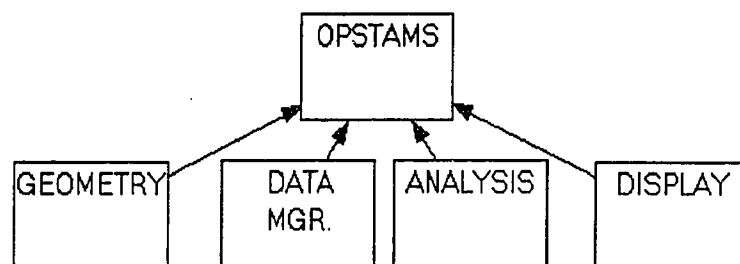


Figure 21. The fundamental abstract OPSTAMS classes.

member classes of these base classes are abstract classes which contain the fundamental data structures and methods which will be combined by the container classes to provide the necessary functionality required by a user. Abstract classes do not have data class members, only member functions. For example, the Raster abstract class (derived from the Geometry class) contains a *draw()* method, but does not contain any individual cells to

draw. These are contained in classes which have been further derived. Any of the class members may be null for either data or methods.

### The Geometry Class

The GEOMETRY class is the cornerstone of the OPSTAMS model. The fundamental derived class (and the true base class) is the POINT class. An instance of the POINT class in its simplest form consists of descriptors in four dimensions of a point: a unique identifier, x, y, z coordinates and a time value in the form of a method (figure 22).<sup>3</sup> The *time()* method is a polymorphic function that returns a single value (for permanently fixed points), a tuple (for points which occupy space over a discrete duration of time), or a function (for points whose existence in time are determined by other factors).

The next fundamental derived class is the LINE class. LINE is derived from the abstract class GEOMETRY. Line uses points, but is not derived from POINT. As in the POINT class, the LINE class has members which are both static data types and methods. A line is minimally defined by an identifier, two endpoints and an optional "curve" method which can be used to define an arc or a complex curve (figure 23). If the line is adequately defined by the two endpoints, the curve method returns a null. The temporality of the line is inherited from the points it uses as defined in the POINT class.

---

<sup>3</sup> Geometry is never stored for its own sake, only when associated with some user-defined object.



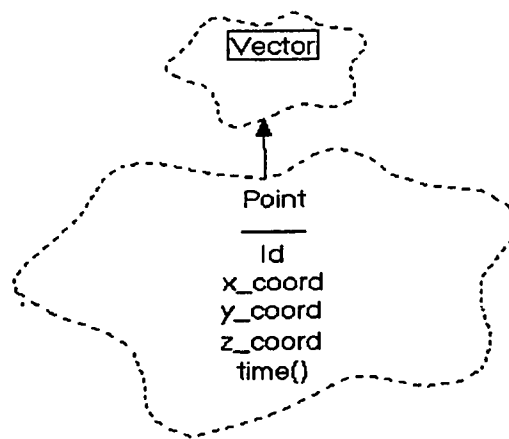


Figure 22. A class member function.

The Line class does not inherit directly from the Point class as one might expect. Rather, it *uses* the Point class. This distinction is most easily described as the difference between the *has-a* and *is-a* relations. A line has two endpoints (which are Point objects), a line is geometry. Therefore, a Line object contains Point objects as class members.<sup>4</sup>

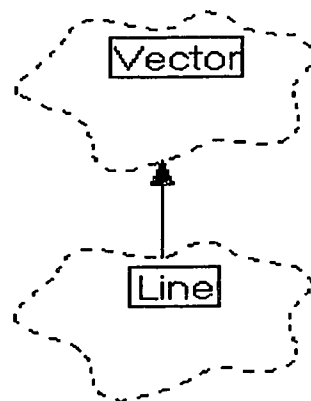


Figure 23. The Line class inherits from the Vector abstract class.

---

<sup>4</sup> The rest of the GEOMETRY class is derived from the Point and Line classes. The complete GEOMETRY class diagram is contained in Appendix A.

### The Data Management Class

Because data is distributed out among the various objects, a standard relational database management system will not adequately support the level of abstraction the OPSTAMS model allows. Unless an object specifically contains an instance of a table or a relation, the table or relation may not exist elsewhere in the system. Each object defined in the model has particular needs with respect to data storage, transformation and manipulation. The Data Management class exists to provide the basic data structures and operations required by specific objects. Objects can inherit the required functionality either from their own immediate superclasses or directly from the Data Management subclasses.

The Data Management class provides many data structures which can be used according to the type of data at hand. Data structures are themselves objects and so can be encapsulated into other objects which in turn can be stored in the object database. Objects are inter-referenced by pointers to the object identifiers (objects containing objects). For example, a road network might be an object stored as an adjacency matrix or linked list. Container classes can be created which hold collections of various objects from different classes. As long as they are contained within an object, they may be stored in the object database.

One of the potential pitfalls of such a schema is that a user may throw everything into one big collection class object. Such an approach would be carrying the ease of data abstraction theme a bit far and would be counterproductive. Queries would be difficult and very slow, and would negate the positive attributes of the object paradigm.<sup>5</sup> As with any data model, care must be taken to organize data so as to optimize the system, given its

---

<sup>5</sup> Throwing everything into one container class is tantamount to assigning all data to a single table in a relational model.

constraints. One of the arguments against object orientation is that it requires careful thought "up front" with respect to class design. This is true, but is necessary in order to reap the benefits of a well organized data model. In general, the greater the level of abstraction a model provides, the easier it is to corrupt.

Object-Oriented Database Management (OODBM). Turnkey object-oriented data management systems have only recently been appearing commercially. This is a relatively new way of doing business as well as representing a paradigm shift away from the traditional relational concept of database management. In the OPSTAMS model, there is a distributed database, in that every object defined in the system may contain data in its own proprietary form. As a result, the data doesn't reside in one physical place such as a table with references to the spatial entities as it would in a traditional relational DBMS. Instead, data is encapsulated directly with each spatial object and is physically located in storage according to the hierarchical organization of the relevant classes. Relationships between objects are handled by pointers to object identifiers.

The structure of class hierarchy and their respective objects is held statically in the storage device through *persistence*. Persistence ensures that what has been created dynamically in memory is preserved accurately in storage. A persistent object is one which is created dynamically and remains static between sessions. For example, if a spatial analysis is performed which results in a new polygon, the polygon is an instance of some class and has data members (and possibly functions) associated with it. It is stored in the database in this fashion, with pointer references to its superclass and members. In successive sessions, the data is available from the object as it was stored. This functionality is contained in the ObjectMgmt class.

Concurrency in Data Analysis. In data management, "concurrency" has been traditionally referred to as a simultaneous request for data from a distributed user base. In the parallel world, the distributed user base may be other objects or processors as well as users. Object locking is the means by which an object's data members are rendered inaccessible while data elements are added, deleted or modified.

A locked object sends messages out to all other objects related to the locked object. Dependencies are thereby updated throughout class hierarchies. Such communication is instigated by an *event*. An event is any operation on or behavior by any object which either changes its own state or that of other objects. OODBMS are considered to be event-driven.

In the case of shared-memory parallelism, operations which access an object's data members could occur simultaneously across all nodes. However, if the data member is modified, all operations would have to go into a wait state until all related objects have been updated. For distributed memory parallelism, it may be preferable to create "copy" objects which are distributed to each of the nodes. After all operations are finished, updates can then be made. Such an approach is best suited to MIMD configurations.

### The Analysis Class

The analysis class provides a minimum level of functionality to perform spatial analysis and data transformations. It is expected that a user would build upon the framework offered in this class to tailor applications for specific analytical requirements. New classes can easily be derived and augmented as needed (see below).

The three base abstract analysis classes are Spatial, Aspatial and User. The Spatial class contains operational objects which are applied to objects residing in the Geometry class. Aspatial operational objects are applied to data members of any class. For instance,

the transformation class would contain routines for performing coordinate transformations between map projections.

The user class is provided as a placeholder for user-defined operations. This is where the flexibility to the OPSTAMS model is utilized. A user can inherit from any of the existing classes in the formation of new, application-specific classes. There is no limit to the extensibility of the user class. Although user classes can exist in the other OPSTAMS base classes (GEOMETRY, etc.), it is the Analysis class where most of the class expansion is intended to take place. A user can inherit from any of the other classes in the model as needed in support of analysis or display.

#### The Display Class

The Display class (Figure 24) contains the methods that can be applied to various spatial objects in order to support visualization and best represent the data on the screen (or other graphic output device). These methods include basic screen operations, ray tracing, foreground/background operations, perspective and multi-media effects. The Display class is comprised mostly of member functions, rather than data. This class is highly dependent on the other main abstract classes (Geometry, Data Management and Analysis), especially when animation is employed.

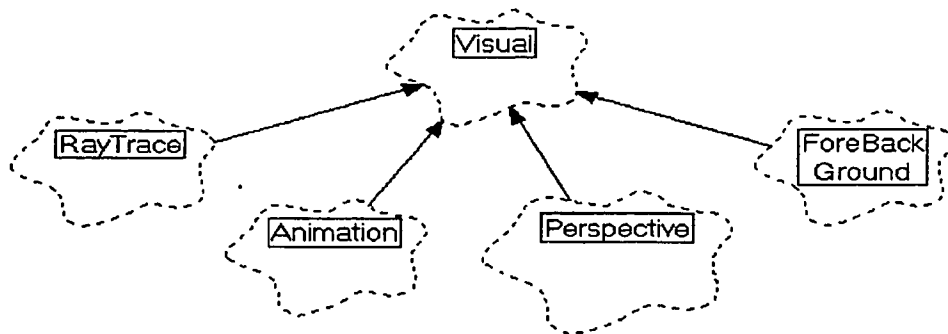


Figure 24. The Display class.

The Display class uses many objects from the Geometry, Data Management and Analysis classes. Geometry objects can be used by the *display()* member function to display the objects. Data structures originating in the Data Management class are used by the Display class member functions to properly organize data for display. In the case of query-based animation, objects returned from a query over an epoch will result in the need for the chronological ordering of the objects for display.

Another example of this interdependency is ray tracing. Ray tracing is geometrically dependent on perspective as the viewpoint moves through space and time. It doesn't matter if a Geometry object uses a Display class member function or the Display class member function uses the Geometry object, the result is the same. This is one of the ways in which the object-oriented paradigm eases problem abstraction. Polymorphism allows the *display()* function (for example) to be used by other classes.

Conversely, the *display()* member function can be used in the Geometry subclasses by objects so that they can display themselves on the screen. It is likely that a user would encapsulate Display class methods in new, application specific classes. For instance, a user-defined class such as Graph (containing a road network) might use the *Display()* member function from the Visual abstract class to draw the road network.

### Polymorphic Functions

Polymorphism and encapsulation are the most powerful concepts of the object paradigm. These provide the greatest contribution to the ease of problem abstraction to the OPSTAMS model. Most of the abstract base classes in the OPSTAMS model contain methods which are inherited by their respective subclasses. The Visual abstract Display class contains the *Display()* member function, the Analysis class has the member functions *GetData()* and *PutData()* contained in the abstract classes Spatial and Aspatial. These functions are inherited by other classes which need their functionality and are modified to

the specific needs of the inheriting class. The best example of this is the *Display()* function.

The *Display()* function originates as a member function in the abstract Visual class from the Display hierarchy. We have seen how a Geometry object can inherit the *Display()* function in order to display itself on the screen. Slight changes are made to the function as it appears in the Line class from how it appears in the Point class. User-defined classes can also take advantage of the *Display()* function, as needed. Because of polymorphism, much of the original code can be reused through an application. Additionally, the user (application developer) does not need to keep track of many names for the same conceptual operation. It is always *Display()*.

### Decomposition

For simplicity, decomposition in the OPSTAMS model is limited to the domain and control decomposition techniques. These two techniques are adequate for the majority of needs in a spatial analysis and display system. Much of the domain decomposition is inherent in the object class hierarchy. An example of this would be the nodes and edges of a graph object. Control decomposition can be applied to procedure-oriented objects such those associated with perspective and ray tracing in visualization.

The manner in which decomposition is executed is dependent on the parallel architecture being utilized. Typically, domain decomposition will most often be employed with SIMD configurations while the control decomposition techniques are better suited to MIMD architectures. Domain decomposition is easier to do "on the fly" and can be automated to be performed at execution time. Control decomposition, on the other hand, needs to be more carefully executed. Individual algorithms need to be optimized for parallel execution.

### System Level Functionality

The organization of the model is to support modularity in application development. This approach is similar to the toolbox approach found in the current generation of GIS, but differs in that this approach takes advantage of late binding and dynamic linking, while the current generation is constrained by early binding. In the current generation, a system is compiled by the vendor and distributed to users. The users are provided with a macro language through which they can manipulate the system to provide analytic functionality. One of the drawbacks to this approach is that interpreted commands are slow in application settings.

Generic GIS are designed to provide the most functionality to the most users, and most users are not very sophisticated at this time. There is a small group of users who *are* sophisticated, and it is this group which usually performs the application programming for the less sophisticated users. These applications are typically designed around a limited problem space and the user is typically trained to use the application, not to be an expert in GIS and spatial analysis per se.

Resulting applications are therefore carrying around excess "baggage" in terms of unutilized functionality overhead. For example, a dispatch application designed to perform shortest path calculations for emergency vehicle response probably has little or no use for overlay analysis. But, because the generic GIS provides this functionality as part of the package in precompiled form, the dispatch application must "carry" this unutilized functionality. The penalty is usually evidenced by slow application execution times.

Another penalty exacted on application developers by the current generation of GIS is due to relational databases held internally to contain spatial data models. Execution time and storage space penalties are paid because these architectures are either not open, or are highly protected. The reasons are usually twofold: 1) to protect the user from



corrupting the data model, and 2) to prevent competitors from gaining access to the system internals.

The OPSTAMS approach supports an open architecture in that it supplies classes (in raw code form). A developer is able to compile his/her own system by selecting just the functionality required by the problem space from the classes available, and adding new functionality and classes as necessary. While this approach requires a *developer* to be comfortable in an object-oriented programming environment, the advantage to the *user* is a fast, easy to use, streamlined application. Such an approach takes advantage of object modularity, late binding, dynamic linking and code reuse, all of which are good software engineering practices.

#### A User's Perspective

To the user, the OPSTAMS model should provide easy problem abstraction. In order to demonstrate this, we will develop an example of how a user might connect objects from classes to perform an analysis. Suppose that the user wishes to model a bus rider's trip from one point to another. Let us further suppose that the OPSTAMS model has been implemented and provides a graphical user interface that permits a user to create and use objects from classes via network programming. In this environment a user can develop an analysis run by simply connecting icons on the screen which represent classes.<sup>6</sup>

The Analysis class offers an abstract class for users to derive their own classes from: the Analysis::User class. Because most object-oriented programming languages do not allow anyone but the programmer to instantiate a new standalone class, a socket must be supplied so that any user-specified classes can "connect" to the rest of the class

---

<sup>6</sup> Advanced Visual Systems offers such an interface for its UNIX-based software.

hierarchy. The abstract User class facilitates such a connection. The user can create any class needed for analysis under the Analysis::User class.

The example we will use to illustrate this process will be to apply the OPSTAMS class structure to create a class that will allow for the modeling of a bus rider's trip from a given starting point to some arbitrary destination. The relationships between riders, routes and stops are somewhat complicated in that a rider uses multiple stops and possibly multiple routes, a route has multiple stops and multiple riders and a stop has multiple riders and may have multiple routes. Figure 25 illustrates these many-to-many relationships.

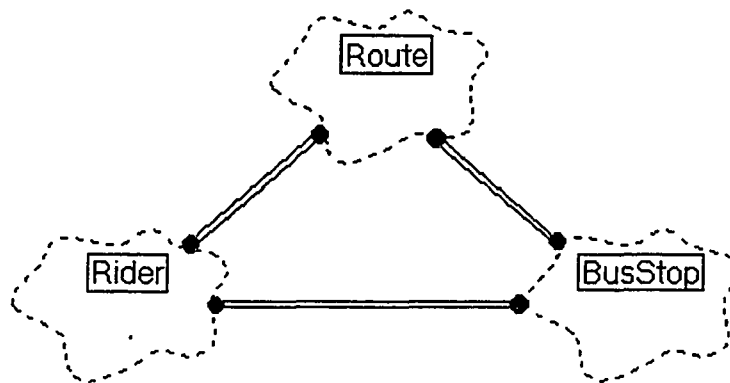
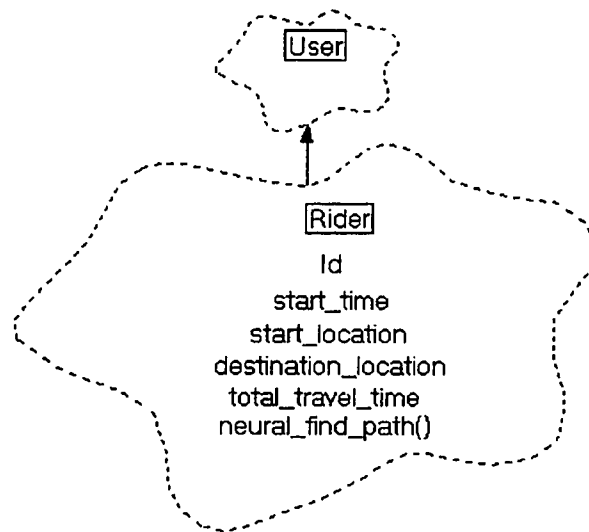


Figure 25. Many-to-many relationships between classes.

In order to perform the analysis, three classes will be derived from the Analysis::User abstract class: Rider, BusStop and Route. Bus stops make good objects because they have data members (location, bus routes, waiting riders, etc.) and behavior (scheduled stops) that can be encapsulated together. The class for bus stops will be called BusStop and will be derived from the User class (because it is user-specified) but will use objects from other parts of the model as well. The Route class will contain route numbers, stop locations and times. Because of the many-to-many nature of the relationships

between Rider, Route and BusStop, these three classes will be created independently of one another as opposed to being inherited (one might think that BusStop should derive from Route, for example).

A menu selection would put the user into "create class" mode. The user can specify what data and behavior the Rider, Route and BusStop classes will need. In this example, the user will specify the rider's starting time and location as well as the destination location and a time constraint for total travel. The Rider class would look something like Figure 26, below. The user would specify which data members of the newly created class were inherited from other classes, or were unique to the new class. The Rider class will contain data members for the rider's location as well as member functions for the analysis. For example, the Rider.start\_location and the Rider.destination\_location would be derived from the Geometry::Point class. Rider.Id and Rider.total\_travel\_time would be new and unique to Rider.



**Figure 26.** The Rider class.

The Route class will contain number, path and stop data members. The Route.number will be a new unique value to the class. Path will be derived from the Analysis::Network class. Thus a Route.path will inherit path origin and destination members from its parent class so the route's originating location and destination can be determined. A Route.path would be a list data structure containing road segments. So that the path can be handled as a single edge in the shortest path algorithm, data members Route.Path\_start and Route.path\_finish would be derived from the path's edge list by a class member function Route.find\_ends(). The Route.stop data member would be a structure containing a scheduled time and stop number.

The BusStop class will contain new data members stop\_number, location, stop\_time, and routes. BusStop.stop\_time is a list of time intervals when the location is active as a stop. For example, if a bus from a given route is scheduled to stop there at 4:00, the stop\_time duration would cover a reasonable interval to account for the bus being slightly early or late, say from 3:50 through 4:10. The BusStop.routes data member would be a list data structure containing Route objects. For example if route numbers 33 and 17 are scheduled for stops at the stop, the BusStop.time values can be determined from the BusStop.route member objects because Route objects have "stop" data members. Because route scheduling is likely to change from time to time, a member function will be added to provide the determination of the stop interval. This member function will be called calc\_times().

Once the classes have been defined, the user can initialize the values for analysis. A menu selection would provide this option. The user might set the Rider.Id to 1, Rider.start\_time to 9:00, click on points of a map on the screen to set the Rider.origin and Rider.destination locations, and enter 2 hrs. for Rider.total\_travel\_time. Via the on-screen network programming interface, the user could connect the rider object to a map object. A menu selection "Analyze" would execute the analysis. When the shortest path has been

found, the `Rider.neural_find_path()` member function will graphically highlight the path on the map on the screen.

In order to perform the analysis, some analytic functionality is needed. In this example, it is provided by using the `Rider.neural_find_path()` member function derived from the `Analysis::Network` class. The `Rider.neural_find_path()` member function is a neural-network based shortest path algorithm that requires origin and destination locations and times in order to execute, but can efficiently handle dynamic network topologies. The times are required because we are modeling bus routes and stops, which can be temporally stochastic, in that their locations are not always bus stops at all times, but rather over intervals of time<sup>7</sup>. Bus routes may or may not exist for a road at a given time of the day.

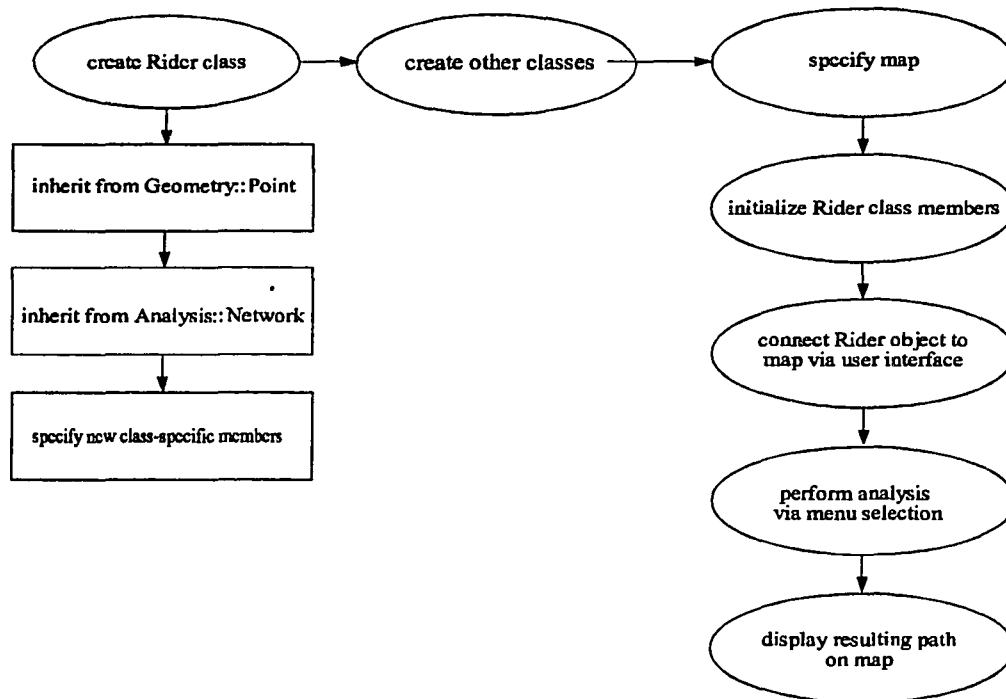
In the real world, bus stops and routes may only be in service for limited times during the day. This correlates to temporally stochastic nodes and edges in a network. In the *neural\_find\_path()* algorithm, bus stops and routes that are not in service at the `Rider.start_time` and `Rider.start_time + Rider.total_travel_time` are excluded from the analysis. This happens because the *neural\_find\_path()* algorithm builds its adjacency list from nodes (BusStop objects which contain Route objects and vice versa) contained in the map specified by the user.

The algorithm would search the node objects (from which the BusStop objects are derived) and edge objects (from Route) of the map and examine each of their respective *time()* member functions for compatibility with the analysis. Only nodes and edges whose times are "true" for the query on the specified time interval (`Rider.start_time` → `Rider.start_time + Rider.total_travel_time`) are included for the analysis. This query ensures that the nodes and edges used in the analysis represent active routes and stops

---

<sup>7</sup> Bus stops could be specified as candidates explicitly, however, the *time()* member provides candidacy implicitly and therefore more robust.

being active. The actual user process of creating the Rider, BusStop and Route classes and performing the analysis is illustrated in the process flow diagram in Figure 27.



**Figure 27.** Process flow diagram for the Rider example.

The advantage the OPSTAMS model has over traditional approaches lies in the ease of abstraction and the application of the neural network based shortest path algorithm. The user needs only to be concerned with the salient features of the problem: the rider, the bus stops and the bus routes. If a route is changed, no additional changes need to be made to the model because the data members are either shared or inherited by the other objects in the model. For example if a new stop is added to a route, it is picked up by the BusStop class because BusStop contains Route objects and stop times are determined within BusStop.calc\_time(). Because the topology of the network is dependent on the riders travel time constraints, the neural shortest path algorithm supports

the transitory nature of bus stops and routes.<sup>8</sup> A parallel implementation would provide more rapid solution times than could be had for dynamic network topologies solved by traditional methods.

---

<sup>8</sup> Refer to Chapter 8 for details of the neural shortest path algorithm.

## CHAPTER VI

### ARTIFICIAL NEURAL NETWORKS IN SPATIAL ANALYSIS

#### INTRODUCTION TO ARTIFICIAL NEURAL NETWORKS

Recently neural nets have been successfully applied to many different problems in optimization, pattern recognition, general classification and forecasting. A good application for a neural net is one for which standard procedural approaches either fail (are intractable) or are computationally infeasible. Neural network applications are sometimes referred to as "parallel distributed processing" due to the inherently parallel nature of the networks.

Fundamentally, a neural net is a set of vectors (layers) of nodes, each node from a layer being completely connected to its adjacent layer(s) of nodes via weighted links known as "synapses". In this configuration, one layer serves as the input vector, one or more layers act as "hidden" vectors and one layer acts as the output vector. In Figure 27, the row of nodes labeled  $i$  is the input layer. The row of nodes labeled  $j$  is the "hidden layer", and the row of nodes labeled  $k$  is the output layer.<sup>1</sup>

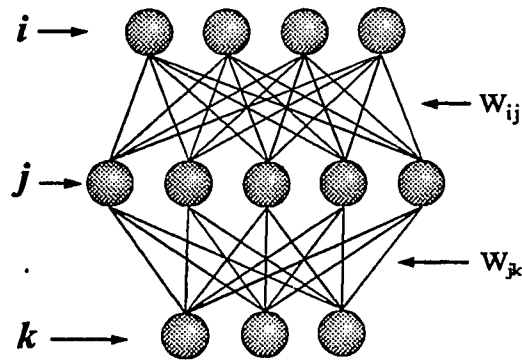
The middle layer is referred to as the hidden layer because only the input and output layers are visible, that is, data is accessed directly only through those two layers. Transformations are performed on the data in the middle layer by the network. Often

---

<sup>1</sup> There is an implicit connection to computational geometry here in that input nodes represent discriminants, which, in turn, correspond to hyperplanes.

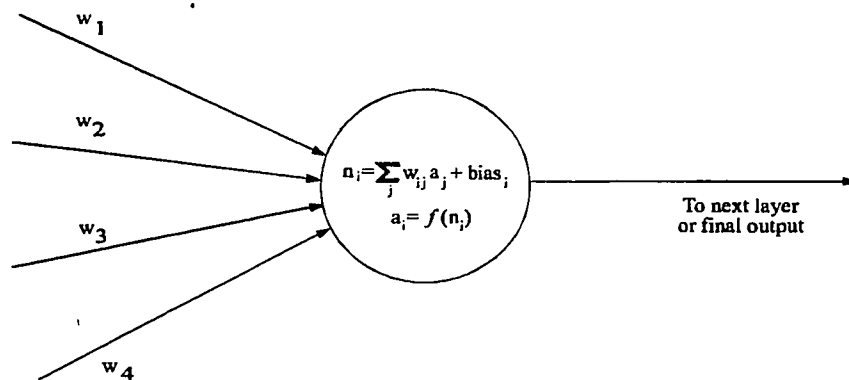


there is more than one hidden layer, depending on the needs of a given model. The connections between the layers are called "synaptic weights" ( $w_{ij}$  and  $w_{jk}$  in Figure 28).



**Figure 28.** A typical neural network topology.

These weights are the transformation functions which amplify or attenuate the outputs of the previous layer's nodes. The net input to each node at the hidden and output layers are the sums of the weighted outputs of the nodes from the previous layer. Typically, the weights are iteratively adjusted across the layers until the errors at the output nodes have converged to some pre-determined limit. This process is known as "learning".



**Figure 29.** Neuron activation.

Nodes in the input layer receive raw data as input. Nodes in the hidden and output layers accept weighted output from previous layers as in Figure 29. Activity at a hidden or output layer consists of summing the weighted inputs, exposing that sum to an activation function and transmitting the product as output. The net summation function ("collection") function is typically

$$net_{pi} = \sum_j w_{ij} a_{pj} \pm bias_i ,$$

where  $net_{pi}$  is collected term for pattern  $p$  at node  $i$  in the current layer,  $w_{ij}$  is the synapse weight between node  $j$  in the previous layer and node  $i$  in the current layer,  $a_{pj}$  is the activation of node  $j$  in the current layer for pattern  $p$ , and the  $bias$  term represents a connection to a node of constant output in the previous layer.

The activation function is of the form  $a_i = f(net_i)$ . Activation functions themselves are usually taken as either the sigmoid function

$$a_{pi} = \frac{1}{1 + e^{-net_{pi}}} ,$$

the unit step function ( $S(x)$ )  $a_{pi} = 0$  for  $net_{pi} = 0$ , 1 for  $net_{pi} > 0$  and -1 for  $net_{pi} < 0$  , or its integral, the unit ramp function ( $R(x)$ )  $a_{pi} = 0$  for  $net_{pi} \leq 0$ , and 1 for  $net_{pi} > 0$ .

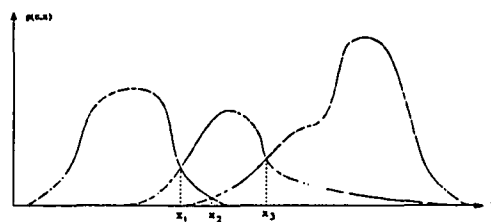
### Discriminants and Hyperplanes

The basic statistical concept that sets the stage for successful application of neural networks is Bayes' decision rule.<sup>2</sup> Simply stated, Bayes' decision rule says that  $X$  belongs to class  $C$  if and only if  $g_j(x_k) > g_i(x_k)$ , where  $g_i(x_k)$  is the decision function for class  $i$  (for

---

<sup>2</sup> A complete treatise on Bayesian statistics is beyond the scope of this dissertation.

all  $i = 1, \dots, n; i \neq j$ ). The products of applying Bayes' decision rule are called the discriminants. The discriminants are determined by probability distributions of  $X$  falling in class  $C$ . In Figure 30, the three bell curves correspond to the probability of  $X$  belonging to  $C_1, C_2$  or  $C_3$  (or,  $p(C, X)$ ).



**Figure 30.** Discriminants

Each intersection of the probability curves indicates a threshold number corresponding to a point on the horizontal ( $x$ ) axis. Each threshold number is a discriminant. In Figure 29, the discriminants are  $x_1, x_2$  and  $x_3$ . The above example is two-dimensional. In  $n$ -space, the discriminants are hyperplanes that split up the pattern space into attribute volumes. Hyperplanes are two-sided, and so can be thought of as  $n$ -dimensional "curtains" between clusterings of patterns based on similar attributes.

### Learning

Networks are "trained" by assigning the proper results to the output nodes and then letting the hidden layer "settle" into a set of weights which will yield the correct output when presented with new data. This process is known as "supervised learning" or "training by example". The imposed solution is then removed from output layer and the network is ready to accept new data. Once the network has been satisfactorily trained it can perform its job without any further supervision. The supervised learning function is given by

$$\Delta w_{ij} = A a_i [c_j - b_j],$$

where  $w_{ij}$  is the synapse weight between a pair of nodes in layers  $i$  and  $j$ ,  $A$  is the learning rate,  $a_i$  is the activation of the  $i^{\text{th}}$  node in one layer,  $c_j$  and  $b_j$  are the target and actual activations of the  $j^{\text{th}}$  node in an adjacent layer.

Another training approach is called "unsupervised learning" or "self-organization". In unsupervised learning, weight adjustments are not made by comparison to some target output, but rather after each iteration, one node is selected as the "winner" and the other synapse weights are adjusted accordingly. The unsupervised learning function can be represented by the equation

$$\Delta \omega_{ij} = \min (\alpha \lambda_i \lambda_j),$$

where  $\omega_{ij}$  is the synapse weight between nodes in layers  $i$  and  $j$ ,  $\alpha$  is the learning rate,  $\lambda_i$  is the activation of the  $i^{\text{th}}$  node in one layer and  $\lambda_j$  is the node activation in an adjacent layer.

Not all neural networks allow for training, however. The Hopfield network model (below) simply converges to target errors which are provided to it. The same network may be reused many times for different objectives by simply changing the target errors of the individual nodes.

### Neural Models

The two most common neural network models are the back-propagation (backprop) network model and the Hopfield network model. Of these, the most widely applied model is the backprop model, followed closely by the Hopfield model and its derivatives. Backprop models are typified by multiple-layered networks while the Hopfield variety are usually single-layered. There are several other popular models, but they are usually some variant of these two basic models. Because these models are the

two most commonly applied, it stands to reason that they are the most likely candidates for applications in spatial analysis.

Backpropagation Networks. The architecture of backpropagation networks is typified by Figure 27.<sup>3</sup> Although they usually consist of three layers, several hidden layers can be used, depending on problem complexity. It has been argued that the three layer model is sufficient for solving even the most complex pattern spaces [Pao89]. Backpropagation networks are very good at pattern recognition and image classification [Hee92]. This model requires training, which can be very time consuming. Additionally, due to the highly iterative nature of neural network algorithms, they are usually very time-costly to operate.

The term "backpropagation" is short for "backpropagation of error". This refers to the notion of introducing data to the first layer of nodes, applying the synapse weighting so as to propagate an error at the second layer. The second layer then sends the "activated" data through the next set of weighted synapses to the output layer, where the cumulative error is sent back up the network, adjusting the synapse weights along the way. Thus, the error is propagated backward through the network.

The number of input layer nodes is determined by granularity of the input data (number of variables). The number of nodes in the hidden layer is somewhat arbitrary, with the number being generally larger than the number of output nodes. The output layer requires a 1:1 mapping between the number of output nodes and the number of output patterns or classes specified. The objective function is typified by

---

<sup>3</sup> These are sometimes referred to as Generalized Delta Rule algorithms. A complete explanation of the Generalized Delta rule is beyond the scope of this dissertation. For a complete treatise, see [Poa89].

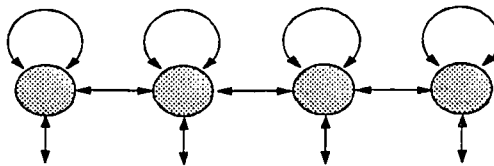
$$\min [\mathbf{E} = \sum_p \sum_i (t_{pi} - O_{pi})^2]$$

where  $\mathbf{E}$  is the error,  $p$  is the vector of training patterns,  $i$  is the vector of output nodes,  $t$  is the desired output (training objective) and  $O$  is the actual activation on a node.

**Hopfield Networks.** Unlike the backprop model, Hopfield nets typically have only one layer. All of the nodes of the layer are interconnected and feed back to themselves as well as acting as the input and output nodes. Figure 31 shows the topology of a typical Hopfield net. Additionally, Hopfield nets are not "trained" but "encoded". Encoding is a process by which a pattern is stored in the net's "memory". This is usually accomplished by a set of vectors such that

$$\mathbf{W}_i = f(\mathbf{A})/\mathbf{R}_i,$$

where  $\mathbf{W}_i$  represents the vector of conductances between the  $i^{\text{th}}$  and  $j^{\text{th}}$  nodes,  $f(\mathbf{A})$  the activation function for the nodes, and  $\mathbf{R}_i$  is the vector of resistances between the  $i^{\text{th}}$  and  $j^{\text{th}}$  nodes. The function  $f(\mathbf{A})$  and the vector  $\mathbf{R}_i$  are user-specified. Specific applications usually require more complex encoding functions, but the above equation serves as a generic foundation.



**Figure 31.** The topology of a Hopfield net.

Recall, the complementary action to encoding, is the process by which a stored pattern is retrieved from memory for comparison. It is accomplished by giving the nodes

some pattern and then allowing the net to "resonate" by sending messages between nodes along the synapses until the net stabilizes at a retrieved pattern. The most common activation function used is the unit step function where  $f(x) = 1$  if  $x > 0$ ,  $f(x) = -1$  if  $x < 0$  and  $f(x) = x$  if  $x = 0$ . The resistance vector  $\mathbf{R}$  is highly application specific and usually requires considerable tweaking in order to elicit the proper responses from the net.

Because of the user-specified activation function and resistance vectors, Hopfield nets are ideally suited to problems rooted in combinatorial optimization such as shortest path calculation, and locational optimization. The example presented in the next chapter utilizes a variant of the Hopfield net and illustrates how a neural network can be used to solve a single pair shortest path problem.

### Neural Nets and the Object Paradigm

Neural networks fit nicely into the object paradigm at two levels. At the first level, each layer of nodes is contained in a vector, which is treated as an object. Synapses between layers are represented by arrays and can be treated as matrix objects. Each vector or array has a general operation it performs and passes data into and out of itself. All nodes can be considered instances of the vector objects (i.e., a unary arrays). Each node has an encapsulated set of data and methods on which it operates. The same holds true for individual synapses. A synapse set between two layers is a matrix object. Each individual synapse ( $w_{ij}$ ) is an instance of the object with a specific reference to a location in the matrix. Because layers and synapse matrices are objects, messages can be passed to them to update themselves.

At the second level, a higher level of abstraction can be had with respect to programming. Class hierarchies can be constructed which reflect the natural organization of the components of a network. For instance, a matrix class derives from a vector class and so on. If the programming language being used to implement the net supports

operator overloading, a shorthand can be devised so that otherwise complicated matrix operations can be simplified, allowing the user to add vectors and matrices as simply as numbers.<sup>4</sup>

### Neural Nets and Parallelism

Neural networks inherently have a great deal of parallelism. A typical decomposition to a SIMD architecture is to treat each layer of nodes as a vector, assigning each node to a processor in the vector of processors. The adjustments to the synapse weights and node activations can be performed in parallel. When all nodes have finished their respective operations, the next layer can be loaded into the processor vector for processing. This process continues until all of the nodes have converged to their respective target errors.

The disadvantage of using a SIMD implementation is that there may be significant time-outs when several nodes converge significantly earlier than others. A MIMD implementation can avoid the time-out problem because the first layer nodes which have converged early can instantly move on and begin processing at the next layer level, i.e., the topology is dynamic. For optimization problems that are usually approached with some form of Simplex algorithms, inherently parallel neural approaches offer more promise for rapid solutions. Simplex routines are difficult to parallelize, and are best suited to conventional sequential implementations [Blu92].

---

<sup>4</sup> Operator overloading is supported by object-oriented languages such as C++. A class operation is assigned to an operator such as "+". When the operator is used with members of the class for which the overload is defined, the operation is executed as specified in the class code. For example, if the "+" operator is defined as being overloaded for the Point class two instances of Point, pt1 and pt2 can be added by specifying  $pt3 = pt1 + pt2$ . This is an obvious improvement over  $pt3.x = pt1.x + pt2.x$  and so on for y and z.



### Neural Nets in Spatial Analysis

The first applications of neural networks to appear in the GIS/Remote Sensing literature were applications of pattern recognition in image processing via raster GIS (see [Fit91] and [Hee92]). Not much appears in the literature to date with respect to applications in vector-base systems. There are many non-graphic and non-spatial applications for pattern recognition based algorithms in urban analysis. Backprop models could be developed for finding principal components in highly auto-correlated data sets such as those used in hedonic pricing models. There is great potential for utilizing neural nets in spatial analysis systems for the solution of optimization problems and other Operations Research-based problems. The OPSTAMS model lends itself well to the inclusion of this methodology because of its parallel/object foundation.

## CHAPTER VII

### THE OPSTAMS MODEL APPLIED TO SHORTEST PATH PROBLEMS

#### OVERVIEW

In order to demonstrate how the OPSTAMS model can be applied to problems in the urban setting, the model will be applied to the single-pair shortest directed path problem. The shortest directed path problem, rooted in graph theory, is well known and has many applications in vehicle routing, electrical power transmission, communications, and contributes to the solution of other higher-order operations research and optimization problems such as location allocation.

A network is constructed from nodes which represent places and edges which connect the nodes and represent the cost associated with moving from a given node to an adjacent one along a common edge. The cost may represent travel time, fares, or any other measure. The algorithms applied to the solution of shortest path problems will all handle any cost abstraction that can be represented as a number associated with an edge. In its most simple form, flows are permitted along either direction of an edge, but the problem can be complicated by constraints on direction of travel along an edge, maximum or minimum costs of edges and penalties for traveling a given edge or edges.

#### The Dijkstra Algorithm

The most famous (and still one of the most efficient) algorithm for solving the shortest path problem was given by Dijkstra in 1959 [Dij59]. The correctness of this algorithm is well documented [Cor90]. Dijkstra's algorithm takes a given graph and

decomposes the graph into lists of nodes and edges. Edges with negative costs are not allowed. Adjacency is computed for each node and either stored in adjacency lists or in an adjacency matrix. Given a source and destination node pair, the algorithm then iterates through the node list, examining edges connecting adjacent nodes, seeking the least cost path to any adjacent node. The node at the end of the least-cost edge becomes, in turn, the next "source" node. The algorithm terminates when a path has been constructed to the destination node.

The computational complexity of the adjacency list version of Dijkstra's algorithm (implemented sequentially) is on the order of  $O(n^2)$ , where  $n$  = the total number of nodes on the graph. This complexity is for the execution of the algorithm *after* the adjacencies have been determined. The adjacency finding procedure itself has a time complexity of approximately  $O(ne^2)$ , where  $n$  = total number of nodes on the graph and  $e$  = total number of edges on the graph. So, the data preparation stage of the shortest path finding procedure is more time consuming than the core algorithm.<sup>1</sup> In pseudocode, the complete sequential process is as follows:

```

construct list of nodes and edges with references to endpoints
construct adjacency lists
  for each node
    search edge list for adjacent nodes
    add adjacent nodes to the current node's adjacency list
  for each node (starting at source, construct minimum path)
    search adjacency list for adjacent nodes
    search edge list for minimum cost connected edge
    select endpoint of minimum cost edge for next node
    add node to path list
  if next node = destination
    quit (print path)
  else
    continue

```

---

<sup>1</sup> This time must be accounted for. It is not an accurate measure of the overall time complexity to pre-process the input data off-line and not include that time in the overall solution time.

In the next section, the OPSTAMS model will be applied to the sequential Dijkstra algorithm to illustrate the level of abstraction that can be attained by organizing the structure of the algorithm according to an appropriate class hierarchy. Once an object-oriented version of Dijkstra has been presented, it will be extended with parallelism.

An Object Dijkstra. The OPSTAMS implementation of Dijkstra will directly use classes derived from the four primary abstract classes: Geometry, Data Management, Analysis, and Display. Figure 32 shows the class structure for OPSTAMS Dijkstra.

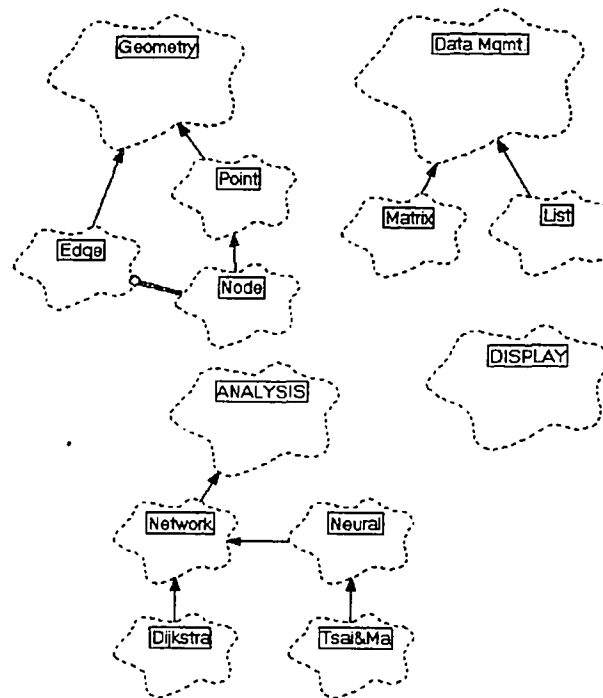
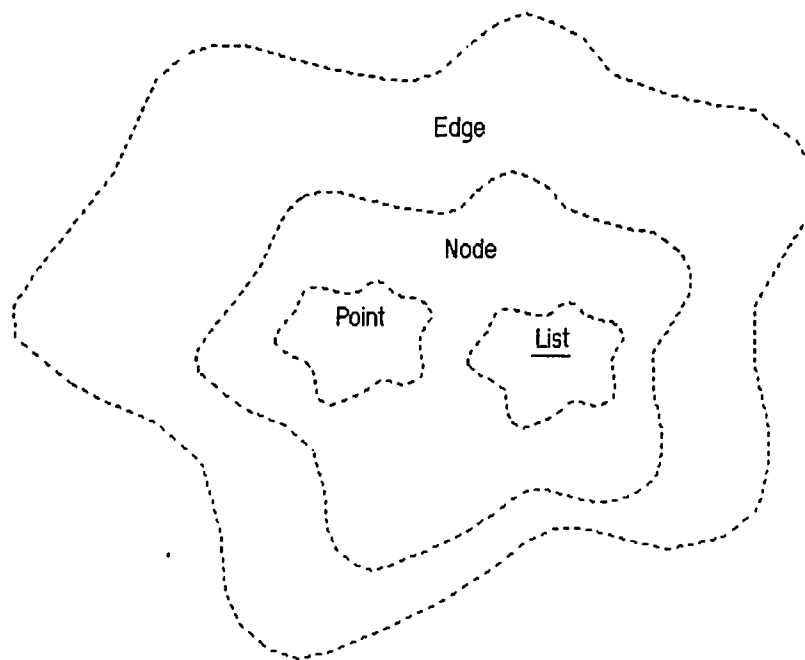


Figure 32. Class structure for object Dijkstra.

Everything begins with the Point class. Point allows for the creation of Point objects. The coordinates for the points which will become the nodes of the graph are

loaded via the *Point.set()* member function. The Node class is derived from Point and so has access to the coordinates contained in the Point objects. The Node class adds its own private data members such as Node.id and Node.label. Each Node object has its own List object for holding adjacent Nodes. Because List is a class which contains its own members and functions, by using List objects within Node objects, Node objects have access to the List class functions like *List.Insert()*. A Node object can add adjacent nodes via the *Node.List.Insert()* member function.



**Figure 33.** Class containment in object Dijkstra.

The Edge class, for illustrative purposes, does not derive directly from the Node class, but is derived from the abstract geometry class and *uses* Node objects as endpoints. This is a subtle distinction, but shows that (in C++) inheritance is not the only way in which objects from one class may reuse code from another class. Indeed, the List in the

Node class has been imported from the Data Management abstract class. Figure 33 shows the Edge class containment hierarchy.

The Analysis abstract class contains the derived class Network. Network has a member function *FindNeighbors()* which uses Node and Edge objects. Because adjacency is an analytical functionality it is contained in the Analysis::Network class rather than in the Node or Edge class. There is no reason why *FindNeighbors()* couldn't belong to the Node class other than it is more consistent to the OPSTAMS model for adjacency to belong to an Analysis-derived class. Similarly, the actual Dijkstra algorithm is a member function of the Analysis::Network class as well. Finally, after the *Network.Dijkstra()* member function has discovered the shortest path between two given nodes, the resulting path can be listed or displayed graphically via member functions of the Display class.

There is now a higher level of abstraction available to the programmer/user, in that all of the messy details such as nested for-loops are included in class member functions. In order to put together a shortest path run, the pseudocode below is all that is needed.

```

Enter origin and destination node numbers
Node.set()
Edge.set()
Network.FindNeighbors(Node[], Edge[])
Network.Dijkstra(Node[], Edge[])
Network.display_path()

```

In turn, this abstraction could be abstracted to the next higher level by encapsulating the above pseudocode into a member function *ShortPath()* which belongs to another Analysis-derived class Graph. The entire process could be invoked through a simple command *Graph.ShortPath(Graph, Node, Node)*. In fact, this level of abstraction is exactly what is needed for the analysis of topologically dynamic networks.

Recall from Chapter 6 that one of the member functions of the Point class is the *Point.time()* member function. For simplification, assume that *Point.time()* returns a 1 or

a 0, corresponding to whether or not the node exists at some time  $t_0$ .<sup>2</sup> Additionally, we are only interested in calling *Network.Dijkstra()* for that static point in time,  $t_0$ . All that is required of the user is to specify time  $t_0$  and the member function will return the shortest path using only the nodes that exist at  $t_0$ . Now the algorithm can be run by calling *Graph.ShortPath(Graph, Node, Node, time)*.

Because time is a local variable, the member function call could be put into a loop and iterated over some time interval specified by the loop incrementer. Nodes whose *time()* member functions return 0 for the specified time are null objects and effectively do not exist for time  $t_0$ . All the user needs to specify to perform the analysis is to supply the parameters Graph object (a predefined set of Node and Edge objects), two Node objects (the origin and destination nodes), and a time variable (a static point in time).

So far, the OPSTAMS model has made problem abstraction easier, but there is no improvement in operational performance. The algorithm is still fundamentally an  $O(n^2)$  time complexity sequential algorithm. This approach is fine for small to medium sized graphs (graphs with less than 1000 nodes), however, performance will certainly be a concern when iterating over many points in time with large graphs. It is time for parallelism.

An Object-Parallel Dijkstra. As noted in Chapter 4, organizing an object hierarchy is the first major step in problem decomposition for parallel algorithms. Since that has already been accomplished, the structure of the class hierarchy must be examined to determine if further decomposition is required, and if so, exactly what form best suits the

---

<sup>2</sup> In actuality, this issue is much more complex, especially when dealing with time eras as opposed to static points in time. *Point.time()* could (and probably would) return a function which would need to be handled according to a set of rules.

problem. On one hand, the OPSTAMS model has exercised domain decomposition with the Node and Edge classes, on the other hand, keeping the *Dijkstra()* and adjacency *FindNeighbors()* member functions within the Analysis abstract class allows for control decomposition to be used on highly iterative algorithms.

Although as with the object oriented paradigm, there are many ways to define and organize class structures, there are even more ways to decompose. For simplicity, we will take advantage of the domain decomposition inherent in the Geometry-derived classes and perform control decomposition on the Analysis-derived classes. Recall that the Dijkstra algorithm is an  $O(n^2)$  time complexity algorithm. This is due to the fundamental nature of the algorithm: a doubly nested for-loop. For each node, we loop number-of-edges times. This is a prime candidate for control decomposition.

Domain decomposition is first applied by assigning each Edge object to a processing node.<sup>3</sup> Because each Edge object contains two Node objects, and each node object contains its own adjacency list, we only need to pass the edge objects to the processing nodes, the data goes automatically, or is constructed within the nodes behind the scenes (another advantage to object-oriented data abstraction!). The node list is inferred by the edge list.

In figure 34, processing nodes a-d represent a four edge graph. The interconnections between processing nodes are for MIMD interprocessor communication; they do not directly represent edge adjacency. However, because all processing nodes are interconnected and represent an edge, for each edge endpoint, one polling operation (*FindNeighbors()*) to the other nodes return the adjacent nodes and edges. The single polling operation has just eliminated the inner edge finding loop of the original algorithm. As a result, this control decomposition has just reduced an  $O(n^2)$  time complexity problem

---

<sup>3</sup> A processing node is an actual microprocessor in the parallel computer.



to an  $O(2e)+C$  time complexity problem, where  $2e$  represents a polling operation for each edge endpoint node and  $C$  is the interprocessor communication overhead.

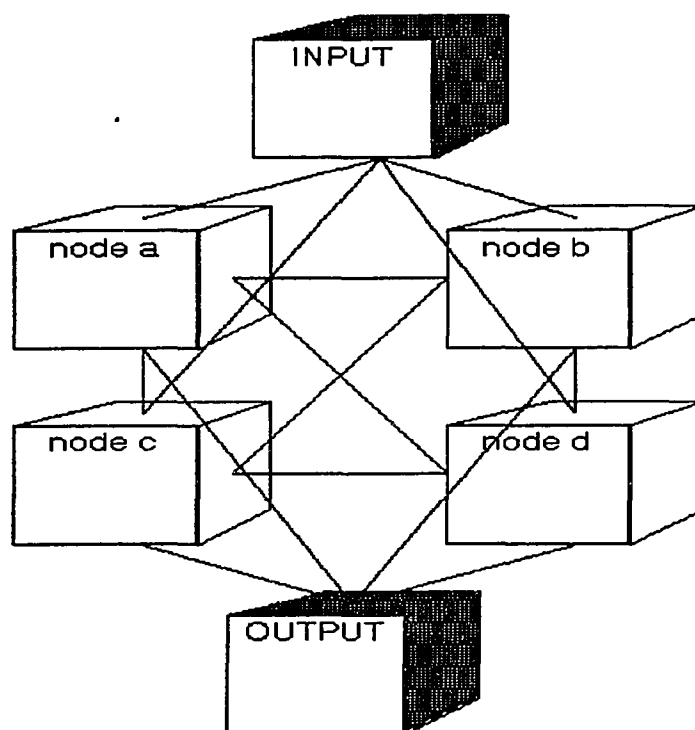


Figure 34. A MIMD topology.

This approach differs greatly from the MIMD Dijkstra implementation of Ding et.al. [Din92] due to the object-oriented nature of the OPSTAMS approach. The approach of [Din92] is more node oriented and requires fewer processing nodes, but more processing steps. This is due to the fact that the architecture they were designing to (a PC-based transputer array) had only a very few processing nodes (4). Their results are very supportive of the OPSTAMS approach in that significant improvements in execution times were realized, even in a node-constrained processing environment. Although parallel Dijkstra algorithms are a step in the right direction, parallel Dijkstra algorithms

may not always be the appropriate tool for the problem at hand. In some cases, an artificial neural network may provide a better way of solving the problem.

### The Tsai and Ma Algorithm

Early attempts at using neural networks for optimization centered around using Hopfield nets for solving the traveling salesman problem [Hop85]. When modified to be applied to all pairs of shortest path problems, these algorithms were plagued by convergence to local solutions, and for several years, it seemed like the application of Hopfield nets would never yield globally optimum solutions. In order to achieve global optima, the basic topology of Hopfield nets were adapted to mechanical analog machines [Mar87(91)]. Actual electronic circuits were constructed to solve the problem. While the circuits did yield globally optimum solutions, the obvious drawback to this approach is that each network to be solved must be physically hard-wired.

One the beneficial results of the analog approach was to display optimization as a function of an energy field or potential surface. Tsai and Ma [Tsa92] approached the problem from the standpoint of finding shortest paths on a network with a dynamic topology, that is, a set of edges that are temporally stochastic. Based on the analog approach of Marcus [Mar87(91)], they adapted the potential surface approach to a neural network topology similar to a Hopfield net. Their approach was to take advantage of the inherent parallelism of neural networks to implement the analog machine used by Marcus.

The results reported by Tsai and Ma show a computational time complexity trend that actually *decreases* as the network size increases.<sup>4</sup> Additionally, the algorithm is more efficient for any network size than sequential Dijkstra. Although Tsai and Ma

---

<sup>4</sup> The reader is directed to the actual Tsai and Ma paper [Tsa92] for charts of this remarkable empirical result.

implemented the algorithm on an analog circuit, they clearly demonstrated the adaptability to a Hopfield-like neural network implemented on a massively parallel computer.

The primary advantage to Tsai and Ma is in its application to topologically-dynamic networks. Applications to vehicle routing and traffic assignment would be an improvement over current methods. Tsai and Ma give the following steps for implementation:

- 1) *Formulate an energy function  $E(\underline{x})=E(x_1, \dots, x_n)$  so that the minimization of  $E$  for  $\underline{x}$  solves the shortest path problem;*
- 2) *Assuming  $\underline{x}$  is available for input, construct a highly parallel feedback circuit that computes all  $N$  gradient components of  $E(\underline{x})$ ;*
- 3) *In parallel, feed the computed gradient components into  $N$  inverting analog integrators whose outputs are then used as feedbacks into the feedback circuit of (2);*
- 4) *Initialize the circuit with problem-specific parameters and wait for the integrators' output  $\underline{x}$  to stabilize;*
- 5) *Retrieve solution to the single-pair shortest path problem using the stabilized  $\underline{x}$ .*

Sequential pseudocode for solving the single pair shortest path problem is as follows, substituting node activation functions for analog integrators:

```

Specify origin and destination nodes
Generate adjacency matrix
Specify energy incrementer and convergence limit
Initialize node activation function5 outputs to 0
Initialize distance(origin node, destination node) to infinity
for Time = 1 to EndOfTime
    for i = 1 to Nodes

```

---

<sup>5</sup> Due to the mathematical complexity of the derivation of the activation functions, the reader is directed to the original text [Tsa92].

```

        initialize node energy to 0
    end
    for j = 1 to Nodes
        increment node energy activation function
    end
    for i = 1 to Nodes
        add taut edges to taut edge list
    end
    assign previous energy level to new energy level
    reinitialize current energy level to 0
    for i = 1 to Nodes
        recalculate aggregate energy level
    end
    if energy < convergence limit
        path found - quit
    else
        continue (increment Time)
    end
end
retrieve shortest path from taut edge list

```

In general, the above algorithm is tantamount to "pulling" the origin and destination nodes in opposite directions. The network can be modeled as a set of strings, each with length proportional to the distances of edges in the adjacency matrix, connected at their respective endpoint nodes. Initially all of the strings (edges) are slack. The network distorts until a straight line exists between the origin and destination nodes. Each edge on the shortest path is considered "taut" and the energy level is stabilized for all of the nodes along the shortest path from the origin node to the destination node. The shortest path is recovered by tracing taut edges from the origin node to the destination node.

The Object-Parallel Tsai and Ma. In order to demonstrate the flexibility of the OPSTAMS model, the model will be applied to the Tsai and Ma algorithm. The algorithm can be implemented on a Hopfield-based neural network by using node activation functions in place of the analog integrators suggested by the authors.

The first step is to perform the first level of decomposition: the class structure. Like the OPSTAMS Dijkstra class structure, the OPSTAMS Tsai and Ma will make use of the Node and Edge classes from the abstract Geometry class, the *FindNeighbors()* member function from the abstract Analysis class, and the List and Matrix classes from the abstract Data Management class. The abstract Display class member functions could be used for display and output.

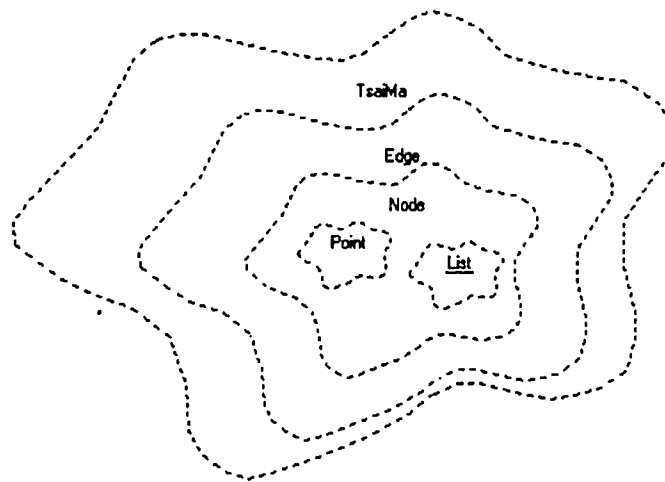


Figure 35. Tsai and Ma class containment.

The fundamental difference between the class structures supporting the two algorithms is at the Network class level. For Tsai and Ma (as well as other neural network algorithms) specialized member functions are needed for node activation and synapse-weighting functions. In order to support these member functions a TsaiMa class is derived from the Network class. In particular, the data members *energy*, *converge\_limit* and the member functions *activate()*, *update()*, and *calc\_energy()* will be created, with *converge\_limit* being added to Network, the rest to Network::TsaiMa.

The next level of decomposition needed is the control decomposition required for splitting up the work among the processor nodes. If the same approach is taken with respect to encapsulation of nodes and edges as was used in OPSTAMS Dijkstra, Edge objects can be assigned to processor nodes on a 1:1 basis. However, additional encapsulation will be needed to capture the necessary activation functions so they are available to each processing node.

Although the original algorithm uses an adjacency matrix, the adjacency list used for Dijkstra will work just as well for TsaiMa since this will be an edge-based implementation. If the adjacency matrix structure is preferred, a member function to construct the matrix could simply be added to the TsaiMa class (or to the Network class from which the TsaiMa class would inherit the functionality).

As stated above, the object-parallel TsaiMa algorithm distributes edge objects across the processing node network. Activation functions are applied to the edges (processing nodes) and to the neural network synapses connecting the processing nodes. The activation function applied to the edges incrementally increases the energy of the edge. Basically this can be compared to incrementing the edge length until the incremented length equals the actual edge length. When this happens, the edge is taut, its energy has stabilized. The synapse activation function is to poll the connected edges (the other network processing nodes) for other stable-state edges. The network "resonates" in this manner until the path is found or the time limit imposed on the run has been reached.

Because of the encapsulated data and member functions distributed across the processor array, the looping evident in the sequential TsaiMa has been reduced to one instruction sent to all of the processing nodes to begin activation. In terms of computational time complexity, the OPSTAMS TsaiMa single pair shortest path algorithm is  $O(1)$  for a 1:1 edge object to processing node topology. With respect to dynamic graph

topology, to change the network, edge objects are either added or deleted from the processor node array and the algorithm is restarted.

There are other neural network approaches to the shortest path problem utilizing backpropagation as well as solutions from computational geometry. Each offer certain advantages depending on the nature of the problem. The highly adaptable OPSTAMS model can pull the appropriate objects and member functions together to provide a high level of abstraction solution to any given problem domain. The strength of the model lies in the simplicity and logical organization of the class structure.

## SUMMARY, CONCLUSIONS AND SUGGESTIONS FOR FUTURE WORK

This final section will examine the OPSTAMS model with respect to the current generation of GIS and to compare and contrast the two approaches to spatial information system design. The most striking difference between the OPSTAMS model and traditional approaches is the use of object-oriented data management as opposed to relational data management. There has been an ongoing debate between database designers for the last several years as to the relative advantages of the object paradigm over the relational paradigm [Atw90]. A few words on this debate with respect to the OPSTAMS model are in order.

The current generation of GIS has three separate primitive modules: the graphic primitive module, the analytic primitive module and the attribute manipulation primitive module. These modules each require interfaces to the others so that data can be extracted, manipulated and displayed. The encapsulation feature offered by the object model eliminates the need for the inter-module interfaces because all three primitive types can be contained by a single object.

The fundamental difference between RDBMS and OODBMS is that RDBMS can be thought of as record-based while OODBMS are considered to be concept-based. RDBMS have data models while OODBMS have conceptual models. The relational model relies on tables joined conceptually by foreign keys. Attributes of entities are fields in the tables. Complex relationships are difficult (if not impossible) to model because they require many join operations. In a large relational data model, a user must have a structure chart on hand just to know where to look for the attributes of interest in order to begin to perform the many two-at-a-time join operations between tables.



Because OODBMS are conceptually organized, complex relationships are the fundamental structures of objects. A class browser is all that is needed to identify data members of different classes, and the simple definition of a new class is all that is required to perform a "logical join" that may require many iterations of two-at-a-time join operations in the relational model. For example, the relational join operation is the culprit behind the awkwardness of the overlay operation in the current generation of GIS. If a user needs to combine different themes into one map, only two maps may be overlaid at a time, forming a third map which in turn may be overlaid with the next map, and so on until the final map product has been constructed.

The reason for the "two-at-a-time" constraint in the current generation of relational-based GIS is that both the spatial data and attribute data models are held in relational databases. There is no escaping the two-at-a-time logical join for either spatial data or attribute data. For example, suppose a user has three different themes (or layers) of information that needs to be combined into a single theme. Let us suppose the themes are parcels, roads and waterbodies: all spatial data. The user would first combine parcels and roads into a temporary table of spatial data, "parcel\_roads". Next parcel\_roads is combined with waterbodies to form the final map. The parcel-roads map can then be deleted because it only served as a temporary map to be combined with waterbodies to produce the final map. The spatial data is held internally in current GIS by relational tables. It is not possible to combine parcels, roads and waterbodies in a single step.

The same holds true for attribute data. In order to perform an attribute-based join such as might be desired for the parcel map, the user might have parcel owner information in one table and parcel tax and zoning information in another table. If the user wished to create a single table containing only the parcel's owner name, parcel size and assessed value, a relational join would be performed on the owner table and the tax and zoning table using owner name from the owner table and size and assessed value from the tax and

zoning table forming a new table "parcel\_own". Again, only two tables at a time can be joined to produce a third.

Any data reduction or aggregation operation in the current generation of relationally-based GIS must iterate over "two-into-one" until the final product has been constructed. Under the object paradigm, a new class can be created in one operation that includes as many other map objects as the user needs into one class in order to compose the desired map product.

Another major difference between the relational and object paradigms is the issue of the separation of code and data in the relational model versus the ability to encapsulate code and data in the object model. The current generation of relational-based GIS require the user to make queries of the database and store the results in a temporary table. In order to use the data for analysis, the user must chain together commands via some macro language. The macro is executed by the GIS interpreting the commands and then exercising the instructions on the temporarily stored data. In the object model, the data, query mechanism and analysis code are encapsulated into a single object. The code and data have been compiled together as an object, so execution is more rapid because the macro language/interpreted command steps are eliminated.

#### Advantages of the OPSTAMS Model

The fundamental advantage of the OPSTAMS model over the current generation of GIS is the ease of problem abstraction to the user offered by the object-oriented approach. The user can create classes that closely resemble the real world situation being modeled. Because the member functions are compiled into the objects, execution is faster than with interpreted commands operating on data held in temporary tables.

Late and dynamic binding offers advantages in that polymorphism may be utilized, further aiding problem abstraction. Late binding allows the program to make the

determination of which form of a command to use. For example DrawPoint, DrawLine and DrawPolygon functions are not needed. Only a single polymorphic Draw command is necessary. A user is faced with fewer commands to memorize because a single command may have many different executions, depending on which object is concerned.

Because of late binding, when the command is issued, the program "knows" which Draw function to use. This is due to the compiler performing strict type checking at compile time. The compiler parses the code for occurrences of the Draw function in the code and keeps track of which classes use the function. At run time, the program "looks up" the appropriate Draw function, depending on the context in which it is used, i.e. which class is calling Draw, and "binds" it to the current portion of code being executed.

Dynamic linking allows the application program to be spread out across many different modules. As a result, only the kernel of the analysis system needs to reside in memory continuously. As different classes are needed, they are dynamically loaded into memory for use. Dynamic linking is like late binding, only for bigger chunks of code. The OPSTAMS model supports dynamic linking due to its four basic abstract classes. Each of these can be compiled into separate modules to be dynamically linked into the application at runtime. Dynamic linking results in better performance because less core memory is required to run an application. This is in contrast to the current generation of GIS which requires separately-executed modules for the kernel of basic commands, an editing module, a plotting module and so forth.

The OPSTAMS model offers better performance in terms of rapid execution times than the current generation can offer for two reasons. The first and most fundamental difference is that the OPSTAMS model is designed to be implemented on massively parallel architectures. The OPSTAMS model does not explicitly require a parallel implementation, but because of the inherent decomposition for parallelism in the object hierarchy, parallel implementation is relatively painless. Current procedural GIS will need

major rewrites in order to be ported to parallel architectures. Secondly, because the need for relational joins have been eliminated, the OPSTAMS model allows complex spatial analysis to be performed in significantly fewer steps. This also aids in problem abstraction and is a major improvement in usability with respect to the end user.

Because the model is to be implemented on a parallel architecture, the inclusion of neural classes offer the analyst new tools for handling previously unwieldy or intractable problems. Applications involving image processing can be analyzed using neural network pattern recognition techniques to quickly analyze large raster based data. Previously, this type of analysis has eluded conventional GIS and has been dealt with off-line by specialized programs. As analysts become more comfortable with neural analytic techniques, more ingenious applications will emerge and the discipline of spatial analysis will advance.

#### Disadvantages of the OPSTAMS Model

One potential drawback to the OPSTAMS model is that it requires the user to think in a new way. Although the objective of the model is to ease problem abstraction for the user, if the user is a dyed-in-the-wool relationally-oriented analyst, there will be some teething pains in learning to abstract a problem in terms of objects rather than tables of attributes. For users new to spatial analysis such as undergraduate students, adaptation to the object paradigm will probably be intuitive and painless. The model should be appealing to the sophisticated analyst who is solving more complicated problems. Novice users probably do not need the power and speed of the OPSTAMS model and might do better with the current generation of GIS.

Another problem with the OPSTAMS model is that it requires the development programmer to do much more "up front" work in terms of class design for parallelism than

would a conventional procedural application. The ease of problem abstraction and quick execution times to the user is at the cost of greater development time.

The extra cost of development time is not the only additional cost associated with the OPSTAMS model over conventional systems. Because of inheritance and encapsulation, there are greater storage requirements in terms of disk space than is needed by the current generation of GIS. In developing code in C++ for testing certain aspects of the OPSTAMS model during the development of this dissertation, a significant amount of duplication was detected throughout the class hierarchy. In C++, when a class is inherited or used by another, everything in that class shows up in the class it is inherited to or used in. There is no way to limit the inherited data members or member functions. For example, in the neural shortest path example, the adjacency lists in the Node class are duplicated for every instance of Node used in the Edge class. This may not be true of other object oriented languages, but it is a known problem with C++. Because the language is still evolving, future versions may address this problem.

Finally, the worst problem with the OPSTAMS model is the exorbitantly expensive platforms required to implement the model. On the low end, a PC could be expanded with Quadputers<sup>TM6</sup> for approximately \$10,000. Additionally, an extremely large hard disk would be required (on the order of several gigabytes) for an additional cost of \$1000 - \$3000, depending on how large the capacity. The next step up would be to use a Sun SPARC Station<sup>TM</sup> with a parallel coprocessor. There are currently several vendors for these add-ons with prices starting at about \$50,000. Add in an appropriately configured SPARC and the cost of a development platform is around \$75,000. At the top of the scale is a dedicated supercomputer running into the hundreds of thousands of dollars for a system. Until massively parallel computers can be readily purchased for under \$10,000, it

---

<sup>6</sup> Microway, Inc.

is unlikely that implementations of OPSTAMS or similar models will be commercially successful.

#### Other Implementation Issues

The major current constraining factor to a full implementation of the OPSTAMS model to a usable system is the lack of standardization in object-parallel language compilers. Compilers are hardware specific and as of now, no preeminent hardware system has emerged to be supported by an object-parallel compiler. As a result, it is unlikely that we will see an OPSTAMS-like system in general use until the hardware/compiler situation stabilizes.

Recently, Kale and Krishnan from the computer science department of the University of Illinois, Urbana-Champaign, have announced a portable C++ based parallel language called CHARM++[Kal93]. The language supports multiple inheritance, dynamic (late) binding and other salient features of C++ extended to parallel objects. CHARM++ offers developers the option to choose between whether objects are sequential or parallel, depending on the nature of the intended use or algorithm (not all algorithms are conducive to parallelization - some are best left sequential). Additionally, the authors have successfully developed traveling salesman problem solving programs in CHARM++ that have run on a variety of parallel architectures from various vendors.

What is most significant and encouraging about CHARM++ is that developers of spatial information systems who are currently developing in the C language can use much of their existing code (with minor modifications) and quickly move into this advanced development environment. The significance of this to members of the GIS and spatial analysis disciplines is that we might soon be using tools that offer us a higher level of abstraction to ease problem formulation and will benefit from rapid execution times offered by parallelism, which will allow the analysis of larger and more complex data sets.

Finally, due to the previously stated complications and difficulties, it is likely that iterative prototyping will be required to "cook" the model down to its final, implementable form. Because of the large storage requirements of the model, some experimentation will be needed to mitigate the impact of duplicated data. Additionally, some adjustments will undoubtedly be necessary with respect to the classes and their member functions on the parallel architecture. Although portability is highly desirable, the model may need major reworks in order to run on different parallel platforms. An iterative prototyping and testing cycle will allow the model to properly evolve and realize its potential.

### Evaluation

Chapter I suggested that the success of an implementation of these concepts can be measured with respect to the current generation by 1) the ability of the system to model temporal data, 2) the ability of the system to analyze and manipulate large data sets with a minimum of operational steps (eliminate multiple relational join operations), and 3) the ability of the system to operate on a parallel computing platform.

Although the design has yet to be implemented in full, a methodology for evaluation against these measurement criteria must be put forth. Such an evaluation might take the form of a head-to-head competition between a relational/sequential-based GIS and a system developed on the OPSTAMS model. Trial problems rooted in combinatorial optimization, spatial overlay and raster/vector integration could be constructed and presented to competent users of each system, along with the necessary data in raw form such as flat files, USGS DLG and SPOT image data. The users would then implement each problem on the respective systems and compare their experiences.

The expectation for the first criterion, ease of problem abstraction, would be that for the sophisticated analyst, problem abstraction would be much easier on an OPSTAMS based system than it would be for a relational/sequential system. This will be especially

true for problems containing a strong temporal component. One potential drawback is that most users abstract problems in terms of the tools they are familiar with. Most analysts are used to relational-based tools. It will take an open mind and/or some time before the analyst is used to the OPSTAMS approach to begin to conceptualize problems in object-oriented terms. Once this hurdle has been cleared, the expectation would be that the OPSTAMS-based system would be the system of choice to all but the most ardent relational model purist.

The second criterion, increased analytic potential, would most likely show at least a two-fold or better increase in the number of analytic methods available for solving spatial analytic problems. Most of the increase will be due to the OPSTAMS model's ability to encapsulate time (in any scale) to a spatial object autonomously, through the ability to utilize artificial neural networks, and the availability of parallelism for visualization. The expectation would be that problems which are currently intractable would become candidates for solution because of parallelism.

The expectation for the third evaluation criterion, execution speed, is most high. Depending on the algorithm, solution times several orders of magnitude faster than are currently possible can be expected. On the average, minimum speed-ups of two to five times can be expected utilizing the currently available parallel hardware. It is very unlikely that any algorithm would run more slowly when implemented in parallel than would the same algorithm implemented on a single processor.

Overall, the expectation would be that the OPSTAMS-based system would provide a much better platform from which the analyst can operate than is currently available. The model would provide better problem abstraction capability, much greater analytic capability and tremendously greater performance.



## SELECTED BIBLIOGRAPHY

- [Agh90] Agha, G., "Concurrent Object-Oriented Programming", Communications of the ACM, Sept. 1990, vol. 33, no. 9, pp. 125-42.
- [AlT90] Al-Taha, K.K. and Barrera, R., "Temporal Data and GIS: An Overview", Proceedings, LIS/GIS '90, vol. 1, pp. 244-254.
- [Arm89] Armstrong, Marc P., Densham, Paul J. and Bennett, David A., "Object Oriented Locational Analysis", Proceedings, GIS/LIS '89, vol. 2, pp. 717-726.
- [Arm90-1] Armstrong, M.P., "Research Directions for Spatial Decision Support Systems: A Position Paper for the NCGIA SDSS Initiative", NCGIA Technical Paper 90-5, Sept. 1990.
- [Arm90-2] Armstrong, M.P. and Bennett, D.A., "A Knowledge Based Object Oriented Approach to Cartographic Generalization", Proceedings, LIS/GIS '90, vol. 1, pp. 48-57.
- [Asp92] Aspinall, R., "Spatial Analysis of Wildlife Distribution and Habitat in a GIS", Proceedings of the Fifth International Symposium on Spatial Data Handling, 1992, vol. 2.
- [Atw90] Atwood, T., "Applying the Object Paradigm to Databases", Computer Language, vol. 7, no. 9, September, 1990.
- [Bab88] Babb, R.G. (ed.), "Programming Parallel Processors", Addison-Wesley, 1988.
- [Bar92] Barrera, R., Egenhofer, M.J. and Frank, A.U., "Robust Evaluation of Spatial Queries", Proceedings of the Fifth International Symposium on Spatial Data Handling, 1992, vol. 1.
- [Blu92] Blum, A., "Neural Networks in C++", John Wiley & Sons, 1992.
- [Bru92] Brugger, B.P. and Muller, J-C., "Mechanisms of Geometric Abstraction", Proceedings of the Fifth International Symposium on Spatial Data Handling, 1992, vol. 1.

- [Cho92] Chou, H-c. and Ding, Y., "Methodology of Integrating Spatial Analysis/Modeling and GIS", Proceedings of the Fifth International Symposium on Spatial Data Handling, 1992, vol. 2.
- [Coh92] Cohen, Fredric, "An Object Oriented Framework for Large Scale Geographic Systems", Proceedings: Technology of Object-Oriented Languages and Systems Tools 8 (1992), Prentice Hall.
- [Cor90] Cormen, T.H. Leiserson, C.E. and Rivest, R.L., "Introduction to Algorithms", McGraw-Hill, 1990.
- [Cro90] Crossland, M.D., "HydroLOGIC -- A Prototype Geographic Information Expert System For Examining An Artificial Intelligence Application In A GIS Environment", Proceedings, LIS/GIS '90, vol. 1, pp. 225-233.
- [DeC90] De Cola, L., "Multiscale Data Models for Spatial Analysis, With Applications to Multifractal Phenomena", Proceedings, Auto-Carto 9, 1990, pp. 313-323.
- [Den89] Densham, Paul J. and Goodchild, Michael F., "Spatial Decision Support Systems: An Agenda", Proceedings, GIS/LIS '89, vol. 2, pp. 707-716.
- [Dij59] Dijkstra, E., "A Note on Two Problems in Connexion in Graphs", Numerische Math., 1959, vol. 1, pp.269-271.
- [Din92] Ding, Y., Densham, P.J. and Armstrong, M.J., "Parallel Processing for Network Analysis: Decomposing Shortest Path Algorithms for MIMD Computers", Proceedings of the Fifth International Symposium on Spatial Data Handling, 1992, vol. 2.
- [Dow90] Dowers, S., Gittings, B.M., Sloan, T.M., Waugh, T.C. and Healey, R.G., "Analysis of Performance on Parallel Architectures and Workstation-Server Systems", Proceedings, LIS/GIS '90, vol. 2, pp. 555-561.
- [Dut89] Dutton, G., "Modelling Locational Uncertainty Via Hierarchical Tessellation", Accuracy of Spatial Databases, eds. Goodchild and Gopal, Taylor & Francis, 1989.
- [Ede87] Edelsbrunner, H., "Algorithms in Combinatorial Geometry", Springer-Verlag, 1987.
- [Ege89] Egenhofer, M.J. and Frank, A.U., "Object-Oriented Modeling in GIS: Inheritance and Propagation", Proceedings, Auto-Carto 9, 1989, pp. 588-598.

- [Ege92] Egenhofer, M.J. and Sharma, J., "Topological Consistency", Proceedings of the Fifth International Symposium on Spatial Data Handling, 1992, vol. 1.
- [Fit91] Fitch, J.P., et.al., "Ship Wake Detection Procedure Using Conjugate Gradient Trained Artificial Neural Networks", IEEE Transactions on Geoscience and Remote Sensing, Vol. 29, No. 5, September 1991, pp. 718-725.
- [Fry92] Frysinger, S.P., "Interactive GIS for Environmental Decision Support", Proceedings of the Fifth International Symposium on Spatial Data Handling, 1992, vol. 2.
- [Gah88] Gahegan, M.N. and Roberts, S.A., "An Intelligent, Object-Oriented Geographical Information System", International Journal of Geographic Information Systems, 1988, vol. 2, no. 2, pp. 101-110.
- [Gol90] Gold, C., "Spatial Adjacency - A General Approach", Proceedings, Auto-Carto 9, 1990, pp. 298-312.
- [Gol89] Goldberg, D.E., "Genetic Algorithms in Search, Optimization and Machine Learning", Addison-Wesley, 1989.
- [Goo91] Goodchild, M.F., "Integrating GIS and Environmental Modeling at Global Scales", Proceedings, LIS/GIS '91, vol. 1, pp. 117-127.
- [Gor90] Gorlen, K.E., Orlow, S.M. and Plexico, P.S., "Data Abstraction and Object-Oriented Programming in C++", John Wiley and Sons, 1990.
- [Gru92] Grupe, F.H., "Along for the Ride: GIS for Vehicle Tracking and Route Selection", Geo Info Systems, September, 1992, pp.44-47.
- [Haz90] Hazelton, N.W.J., Leahy, F.J. and Williamson, I.P., "On the Design of Temporally- Referenced, 3-D Geographic Information Systems: Development of Four Dimensional GIS", Proceedings, LIS/GIS '90, vol. 1, pp. 357-372.
- [Haz92] Hazelton, N.W.J., "Beyond the 2-D Map: A New Metaphor for Multi-Temporal 4-D GIS", Proceedings, LIS/GIS '92, vol. 1, pp. 303-313.
- [Hea90] Healey, R.G. and Desa, G.B., "Transputer Based Parallel Processing for GIS Analysis:Problems and Potentialities", Proceedings, Auto-Carto 9, 1990, pp. 90-99.

- [Hee92] Heermann, P.D. and Khazenie, N., "Classification of Multispectral Remote Sensing Data Using a Back-Propagation Neural Network", IEEE Transactions on Geoscience and Remote Sensing, Vol. 30, No. 1, January 1992, pp. 81-88.
- [Hei91] Heileman, G.L., Brown, H.K. and Georgiopoulos, M., "Simulation of Artificial Neural Networks Models Using an Object-Oriented Software Paradigm", Proceedings of the International Conference on Neural Networks, 1991.
- [Hil85] Hillis, W.D., "The Connection Machine", MIT Press, 1985.
- [Hoc61] Hocking, J.G. and Young, G.S., "Topology", Dover Publications, Inc., 1961
- [Hop85] Hopfield, J.J. and Tank, D.W., "Neural Computation of decisions in Optimization Problems", Biological Cybernetics, 1985, vol.52, pp. 141-152.
- [Hop92] Hopkins, S., Healy, R.G. and Waugh, T.C., "Algorithm Scalability for Line Intersection Detection in Parallel Polygon Overlay", Proceedings of the Fifth International Symposium on Spatial Data Handling, 1992, vol. 1.
- [Kal93] Kale, L.V. and Krishnan, S., "CHARM++: A Portable Concurrent Object Oriented System Based on C++", draft whitepaper, University of Illinois, Urbana-Champaign, 1993.
- [Kem92] Kemp, Z. and Thearle, R., "Modelling Relationships in Spatial Databases", Proceedings of the Fifth International Symposium on Spatial Data Handling, 1992, vol. 1.
- [Kje86] Kjerne, D and Dueker, K., "Modeling Cadastral Spatial Relationships Using an Object-Oriented Language", publication of Portland State University's Center for Urban Studies, Portland, Oregon, 1986.
- [Kje88] Kjerne, D. and Dueker, K., "Modeling Cadastral Spatial Relationships Using Smalltalk-80", Proceedings, LIS/GIS '88, pp. 373-385.
- [Lak90] Lakshmivarahan, S. and Dhall, S.K., "Analysis and Design of Parallel Algorithms: Arithmetic and Matrix Problems", McGraw-Hill, 1990.
- [Lan89] Langran, G., "A Review of Temporal Database Research and its Use in GIS Applications", International Journal of Geographic Information Systems, vol. 3, no. 3, 1989, pp. 215-232.
- [Lan92] Langran, G., "States, Events and Evidence: The Principal Entites of a Temporal GIS", Proceedings, LIS/GIS '92, pp. 416-425.

- [Leu92] Leung, Y., Goodchild, M.F. and Lin, C-C., "Visualization of Fuzzy Scenes and Probability Fields", Proceedings of the Fifth International Symposium on Spatial Data Handling, 1992, vol. 2.
  
- [Li 92] Li, B., "Opportunities and Challenges of Parallel Processing in Spatial Data Analysis: Initial Experiments with Data Parallel Map Analysis", Proceedings, LIS/GIS '92, pp. 445-458.
  
- [Luk90] Lukatela, H., "Hipparchus Data Structures: Points, Lines and Regions in Spherical Voronoi Grid", Proceedings, Auto-Carto 9, 1990, pp. 164-170.
  
- [Mar87] Marcus, R., "A Connectionist Algorithm for Minimum Cost Network Flows", Proceedings of the International Conference on Neural Networks, 1991, vol. 3, pp. 735-739.
  
- [Mac92] Mackay, D.S., Robinson, V.B. and Band, L.E., "Development of an Integrated Knowledgebased System for Managing Spatiotemporal Ecological Simulations", Proceedings, LIS/GIS '92, pp. 494-503.
  
- [McA90] McAbee, J.L.III and Owen, P.K., "A Methodology for the Integration of Spatial Analysis Technologies", Proceedings, LIS/GIS '90, vol. 2, pp. 598-608.
  
- [Mow92] Mower, J.E., "Building a GIS for Parallel Computing Environments", Proceedings of the Fifth International Symposium on Spatial Data Handling, 1992, vol. 1.
  
- [Mul92] Muller, J-C., "Parallel Distributed Processing: An Application to Geographic Feature Selection", Proceedings of the Fifth International Symposium on Spatial Data Handling, 1992, vol. 1.
  
- [Oat90] Oaten, Gregory W. and Shortis, Mark R., "An Object Oriented Approach to Analysis for Geographic Information Systems", Proceedings of the 1990 Annual Conference of the Urban and Regional Information Systems Association, vol. 2, pp. 244-255.
  
- [Odb92] Odberg, Erik, "What "What" is and isn't: On Query Languages for Object-Oriented Databases", Proceedings: Technology of Object-Oriented Languages and Systems Tools 8 (1992), Prentice Hall.
  
- [Olv90] Oliver, M.A., and Webster, R., "Kriging: A Method Interpolation for Geographic Information Systems", International Journal of Geographic Information Systems, 1990, vol. 4, no. 3, pp. 313-332.

- [Ope88] Openshaw, S., "Developments in Geographic Information Systems", Economic and Social Research Council, Newsletter No. 63, 1988, pp. 11-14.
- [Pal89] Pallas, J.I., "Multiprocessor SMALLTALK: Implementation, Performance, and Analysis, Ph.D. Dissertation, STAN-CS-90-1315, Stanford University, Stanford California, 1989.
- [Pao89] Pao, Y.H., "Adaptive Pattern Recognition and Neural Networks", Addison-Wesley, 1989.
- [Pig91] Pigot, S., "Topological Models for 3-D Spatial Information Systems", Proceedings of Auto-Carto 10, 1991.
- [Pig92] Pigot, S. and Hazelton, B., "The Fundamentals of a Topological Model for a Four-Dimensional GIS", Proceedings of the Fifth International Symposium on Spatial Data Handling, 1992, vol. 2.
- [Pre85] Preparata, F.P. & Shamos, M.I., "Computational Geometry: An Introduction", Springer-Verlag, 1985.
- [Pri89] Price, S., "Modelling the Temporal Element in Land Information Systems", International Journal of Geographic Information Systems, vol. 3, no. 3, 1989, pp. 233-243.
- [Rag91] Ragsdale, S., "Parallel Programming", McGraw-Hill, 1991.
- [Rei91] Reibling, L.A., and Olinger, M.D., "A Neural Network Implementation of Parallel Search for Multiple Paths", Proceedings of the International Conference on Neural Networks, 1991.
- [Rex89] Rex, D.B., "Computational Geometry in Geographic Information Analysis", Proceedings of the Urban and Regional Information Systems Association, 1989, pp. 129-136.
- [Rex91] Rex, D.B., White, M.E. and Millard, D., "Vertical Integration of Spatial Data Objects", unpublished internal white paper for the Bureau of Land Management's ALMRS GIS Prototype project, Battelle Pacific Northwest National Laboratory, 1991.
- [Rum91] Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F. and Lorenson, W., "Object-Oriented Modelling and Design", Prentice Hall, 1991.

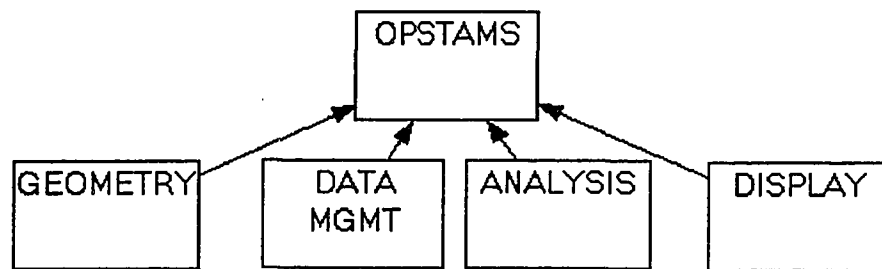
- [Sam90-1] Samet, H., "Design and Analysis of Spatial Data Structures", Addison-Wesley, 1990. (1)
- [Sam90-2] Samet, H., "Applications of Data Structures", Addison-Wesley, 1990. (2)
- [Sch92] Schneider, R. and Kriegel, H-P., "Indexing the Spatiotemporal Monitoring of a Polygonal Object", Proceedings of the Fifth International Symposium on Spatial Data Handling, 1992, vol. 1.
- [Smi92] Smith, T.R., "Towards a Logic-Based Language for Modeling and Database Support in Spatio-Temporal Domains", Proceedings of the Fifth International Symposium on Spatial Data Handling, 1992, vol. 2.
- [Sol92] Solka, J.L., Perry, J.C., Poellinger, B.R. and Rogers, G.W., "Autorouting Using a Parallel Dijkstra Algorithm with Embedded Constraints", Proceedings of the International Conference on Neural Networks, 1992.
- [Stu90] Stuart, N., "Quadtree GIS - Pragmatics for the Present, Prospects for the Future", Proceedings, LIS/GIS '90, vol. 1, pp. 373-382.
- [Sui92] Sui, D.Z., "An Initial Investigation of Integrating Neural Networks with GIS for Spatial Decision Making", Proceedings, LIS/GIS '92, pp. 727-736.
- [Tak92] Takefuji, Y., "Neural Network Parallel Computing", Kluwer Academic Publishers, 1992.
- [Tsa92] Tsai, K. and Ma, R.P., "Artificial Neural Networks for Distributed Adaptive Routing on Dynamic Topology Networks", Proceedings of the International Conference on Neural Networks, 1992.
- [vGr92] van Grol, H.J.M. and Bakker, A.F., "Special Purpose Parallel Computer for Traffic Simulation", Transportation Research Record 1306, 1992.
- [Ver88] Verts, W.T. and Thomson, C.L., "Parallel Architectures for Geographic Information Systems", ASPRS Proceedings, 1988, pp. 101-107.
- [Wan92] Wang, F., "Incorporating a Neural Network into GIS for Agricultural Land Suitability Analysis", Proceedings, LIS/GIS '92, pp. 804-813.
- [Wor90] Worboys, M.F., Hearnshaw, H.M. and Maguire, D.J., "Object-oriented Data Modelling for Spatial Databases", International Journal of Geographic Information Systems, 1990, vol. 4, no. 4, pp. 369-383.

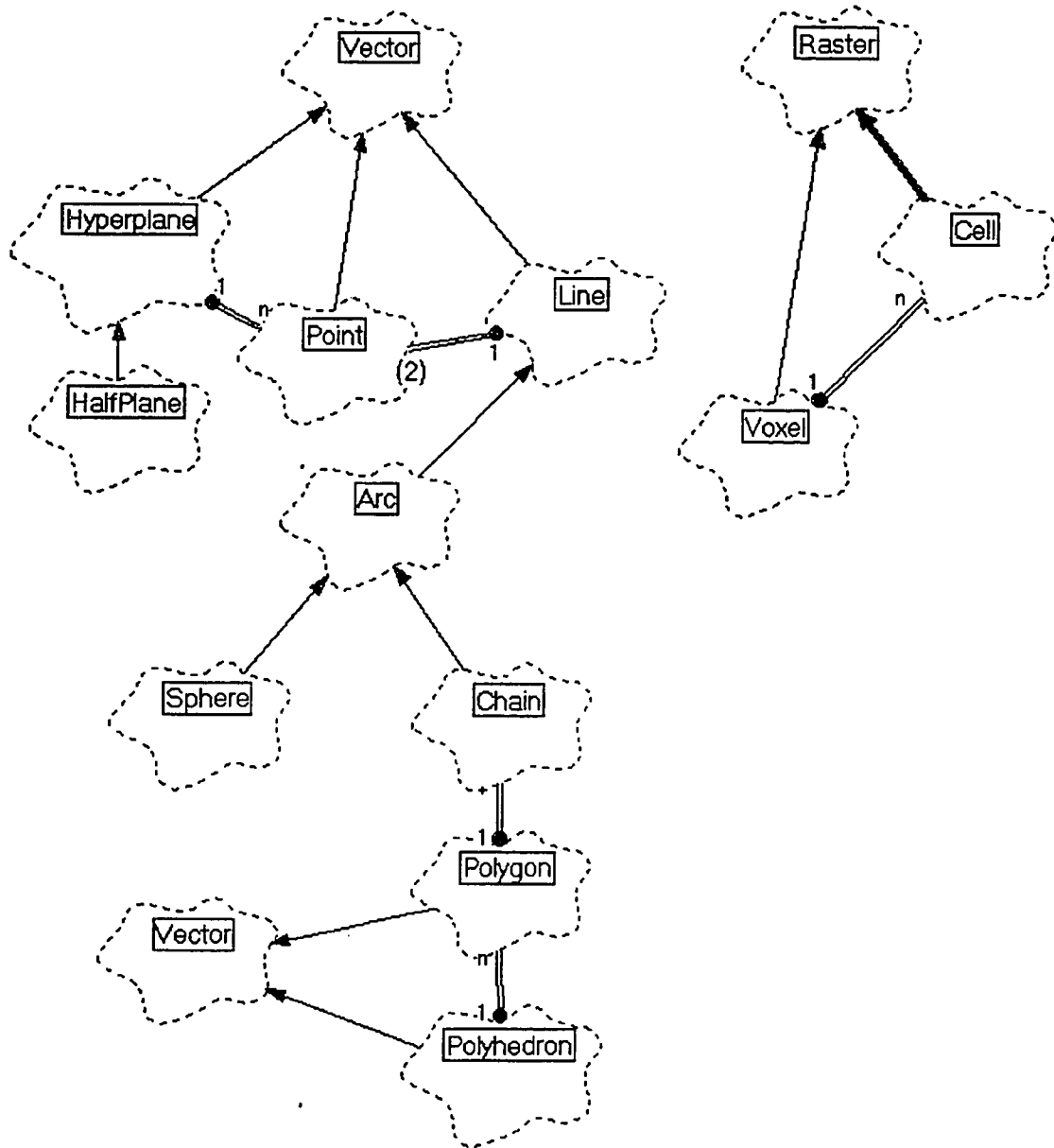
- [Wor92] Worboys, M.F., "A Model for Spatio-Temporal Information", Proceedings of the Fifth International Symposium on Spatial Data Handling, 1992, vol. 2.
- [Yon87] Yonezawa, A. and Tokoro, M. (eds.), "Object-Oriented Concurrent Programming", MIT Press, 1987.
- [Zha92] Zhan, F. and Mark, D.M., "Object-Oriented Spatial Knowledge Representation and Processing: Formalization of Core Classes and Their Relationships", Proceedings of the Fifth International Symposium on Spatial Data Handling, 1992, vol. 2.
- [Zhe93] Zheng, W. and Shanholtz, V.O., "Object-Oriented Programming Opens the Gateway to Real-World Problem Solving", *GIS World*, Vol. 6, No. 1, January, 1993, pp.48-50.



## APPENDIX

Schematic of the OPSTAMS model  
(from Rational ROSE case tool)





**Class name:**

**Point**

**Documentation:**

Point derives vector's virtual member functions.

**Visibility:** Exported

**Cardinality:** n

**Hierarchy:**

Superclasses: Vector

**Public Interface:**

Operations: time()

**Private Interface:**

Fields: id

x

y

z

**State machine:** No

**Concurrency:** Sequential

**Persistence:** Transitory

**Class name:**

**Line**

**Documentation:**

Line "uses" two points. Is derived from vector so it can inherit vector's virtual member functions.

**Visibility:**           Exported

**Cardinality:**           n

**Hierarchy:**

**Superclasses:** Vector

**Uses for Implementation:**

    Point

**Private Interface:**

**Fields:**           id

            begPoint

            endPoint

**State machine:** No

**Concurrency:** Sequential

**Persistence:** Transitory

**Class name:**

**Arc**

**Visibility:** Exported

**Cardinality:** n

**Hierarchy:**

**Superclasses:** Line

**Public Interface:**

**Fields:** id

**Private Interface:**

**Fields:** centerPoint  
radius

**State machine:** No

**Concurrency:** Sequential

**Persistence:** Transitory

**Class name:**

**Chain**

**Visibility:** Exported

**Cardinality:** n

**Hierarchy:**

**Superclasses:** Arc

**State machine:** No

**Concurrency:** Sequential

**Persistence:** Transitory

**Class name:**

**Polygon**

**Visibility:** Exported

**Cardinality:** n

**Hierarchy:**

**Superclasses:** Vector

**Uses for Implementation:**

Chain

**State machine:** No

**Concurrency:** Sequential

**Persistence:** Transitory

**Class name:**

**Sphere**

**Visibility:**           **Exported**

**Cardinality:**           **n**

**Hierarchy:**

**Superclasses:**   **Arc**

**Public Interface:**   **.**

**Fields:**           **id**

**State machine:**   **No**

**Concurrency:**       **Sequential**

**Persistence:**       **Transitory**



Class name:  
Polyhedron

Visibility: Exported

Cardinality: n

Hierarchy:

Superclasses: Vector

Uses for Implementation:  
Polygon

State machine: No

Concurrency: Sequential

Persistence: Transitory

Class name: .  
Vector

**Documentation:**

The abstract base class for all vector-based objects.

**NOTE:** Abstract classes do not have data members, only virtual member functions.

Visibility: Exported

Cardinality: 0

**Hierarchy:**

Superclasses: none

**Public Interface:**

Operations: adjacent\_to()  
part\_of()  
show()  
time()

State machine: No

Concurrency: Sequential

Persistence: Transitory

**Class name:**  
**Raster**

**Documentation:**  
Abstract base class for all raster objects.

**NOTE:** Abstract classes do not have data members, only virtual member functions.

**Visibility:** Exported

**Cardinality:** 0

**Hierarchy:**

Superclasses: none

**Public Interface:**

Operations: show()

**State machine:** No

**Concurrency:** Sequential

**Persistence:** Transitory

Class name:

Cell

Visibility: Exported

Cardinality: n

Hierarchy:

Superclasses: none

Private Interface:

Fields: id

boundary

State machine: No

Concurrency: Sequential

Persistence: Transitory

**Class name:**

**Voxel**

**Visibility:**           **Exported**

**Cardinality:**           **n**

**Hierarchy:**

**Superclasses:**   **Raster**

**Uses for Implementation:**

**Cell**

**Public Interface:**

**Fields:**           **id**

**Private Interface:**

**Fields:**           **boundaries**

**State machine:**   **No**

**Concurrency:**     **Sequential**

**Persistence:**     **Transitory**

**Class name:**  
Hyperplane

**Documentation:**  
Derives Vector's virtual member functions.

**Visibility:** Exported

**Cardinality:** n

**Hierarchy:**  
Superclasses: Vector

**Uses for Implementation:**  
Point

**Private Interface:**

Fields: id

**State machine:** No

**Concurrency:** Sequential

**Persistence:** Transitory

**Class name:**  
**HalfPlane**

**Documentation:**  
**One side of a hyperplane.**

**Visibility:**           **Exported**  
**Cardinality:**           **n**

**Hierarchy:**  
**Superclasses:**   **Hyperplane**

**Protected Interface:**

**Fields:**           **orthoPoint**  
A point orthogonal to but not on the  
hyperplane. The orthoPoint is in space  
on the designated side of the  
hyperplane.

**Private Interface:**

**Fields:**           **id**

**State machine:**   **No**  
**Concurrency:**    **Sequential**  
**Persistence:**     **Transitory**

**Class name:**

**Vector**

**Visibility:** Exported

**Cardinality:** 0

**Hierarchy:**

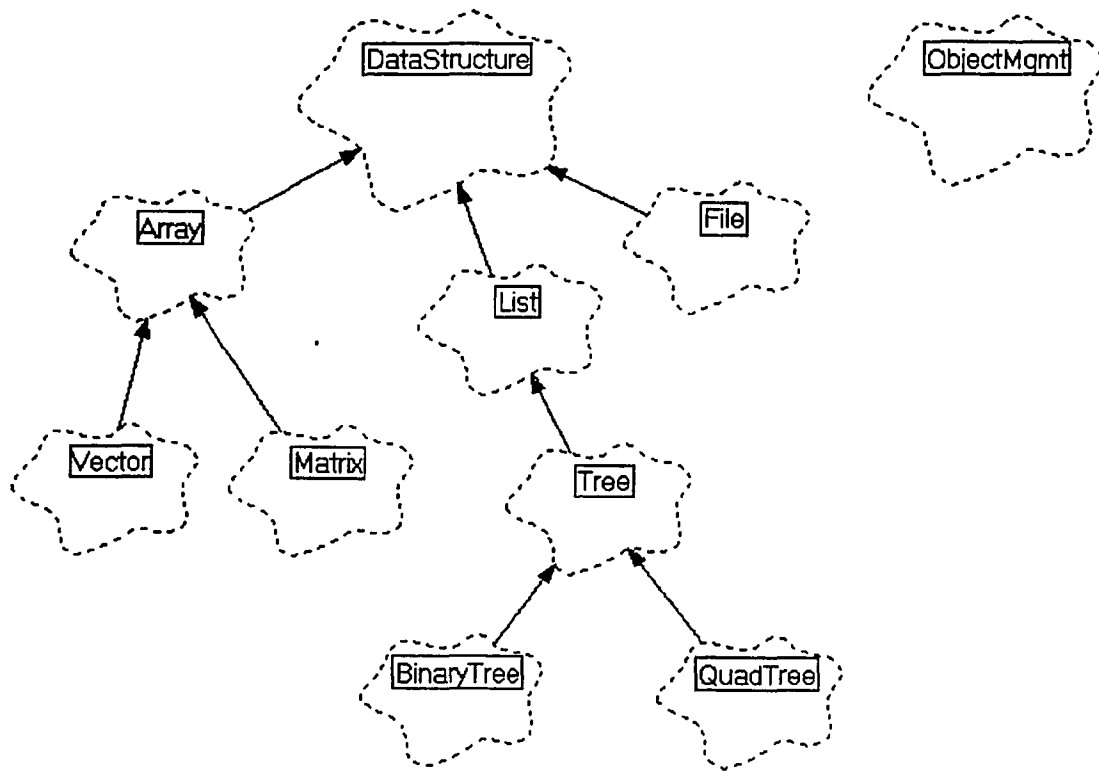
**Superclasses:** none

**State machine:** No

**Concurrency:** Sequential

**Persistence:** Transitory





**Class name:**  
**DataStructure**

**Documentation:**  
    Abstract class for fundamental data structures such as  
    lists, arrays, etc.

**Visibility:**       Exported  
**Cardinality:**       n  
**Hierarchy:**  
    **Superclasses:** none  
**State machine:** No  
**Concurrency:** Sequential  
**Persistence:** Transitory

**Class name:**

**Vector**

**Documentation:**

**Vector data structure.**

**Visibility:**           **Exported**

**Cardinality:**           **n**

**Hierarchy:**

**Superclasses:**   **Array**

**State machine:**   **No**

**Concurrency:**       **Sequential**

**Persistence:**       **Transitory**

**Class name:**

**Matrix**

**Documentation:**

**Matrix data structure.**

**Visibility:**           **Exported**

**Cardinality:**           **n**

**Hierarchy:**

**Superclasses:** **Array**

**State machine:** **No**

**Concurrency:** **Sequential**

**Persistence:** **Transitory**

**Class name:**  
**List**

**Documentation:**  
**List class for linked lists.**

**Visibility:**       **Exported**  
**Cardinality:**       **n**  
**Hierarchy:**  
    **Superclasses:** **DataStructure**  
**State machine:** **No**  
**Concurrency:**   **Sequential**  
**Persistence:**   **Transitory**

**Class name:**

**Array**

**Documentation:**

**Array data structure.**

**Visibility:**       **Exported**

**Cardinality:**       **n**

**Hierarchy:**

**Superclasses:** **DataStructure**

**State machine:** **No**

**Concurrency:** **Sequential**

**Persistence:** **Transitory**

**Class name:**

**File**

**Documentation:**

File input/output class. Methods handle read/write operations and formats.

<b>Visibility:</b>	Exported
<b>Cardinality:</b>	n
<b>Hierarchy:</b>	.
<b>Superclasses:</b>	DataStructure
<b>State machine:</b>	No
<b>Concurrency:</b>	Sequential
<b>Persistence:</b>	Transitory

Class name:

Tree

Documentation:

Tree structure class.

Visibility: Exported

Cardinality: n

Hierarchy:

Superclasses: List

State machine: No

Concurrency: Sequential

Persistence: Transitory



**Class name:**

**BinaryTree**

**Documentation:**

**Binary tree class.**

**Visibility:** Exported

**Cardinality:** n

**Hierarchy:**

**Superclasses:** Tree

**State machine:** No

**Concurrency:** Sequential

**Persistence:** Transitory

**Class name:**

**QuadTree**

**Documentation:**

Quadtree data structure class.

**Visibility:**       **Exported**

**Cardinality:**       **n**

**Hierarchy:**       **.**

**Superclasses:**   **Tree**

**State machine:**   **No**

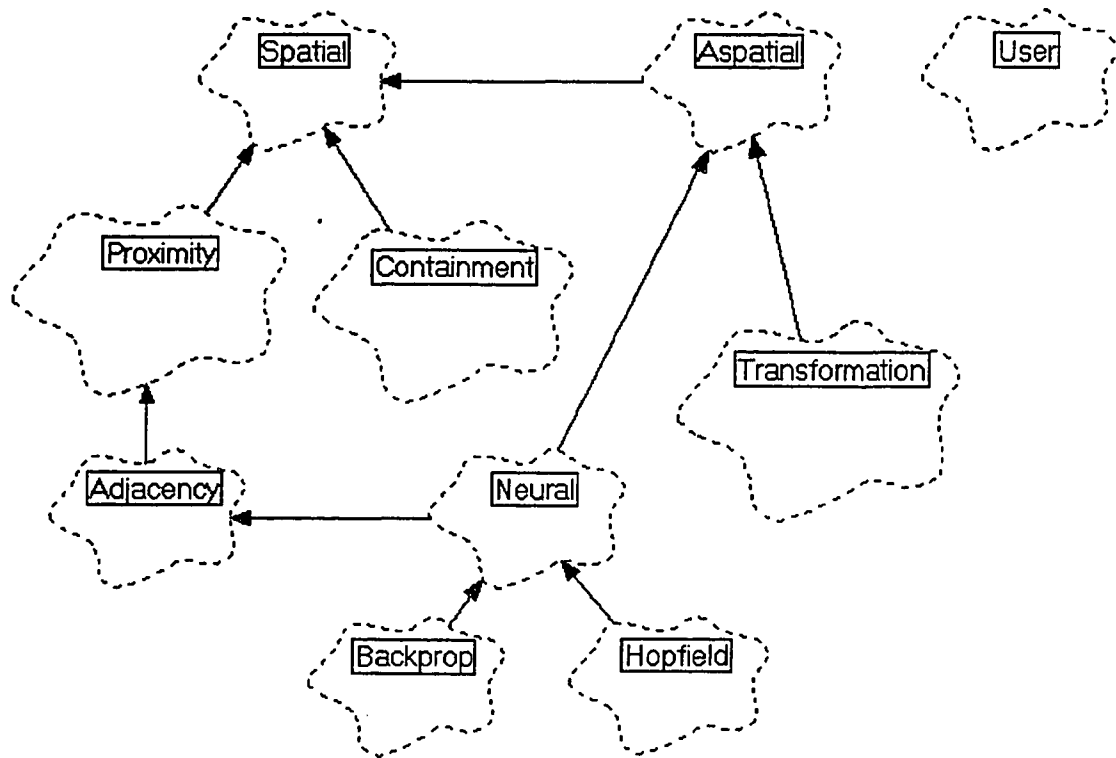
**Concurrency:**     **Sequential**

**Persistence:**     **Transitory**

**Class name:**  
**ObjectMgmt**

**Documentation:**  
The object management class handles the storage,  
retrieval and queries of object data.

<b>Visibility:</b>	<b>Exported</b>
<b>Cardinality:</b>	<b>n</b>
<b>Hierarchy:</b>	
<b>Superclasses:</b>	<b>none</b>
<b>State machine:</b>	<b>No</b>
<b>Concurrency:</b>	<b>Sequential</b>
<b>Persistence:</b>	<b>Transitory</b>



**Class name:**  
**Spatial**

**Documentation:**  
**Abstract analysis class**

**Visibility:**        **Exported**

**Cardinality:**        **n**

**Hierarchy:**

**Superclasses:**    **none**

**Public Interface:**

**Fields:**        **GetData()**

**Receives data from input object.**

**PutData()**

**Sends modified data to output object**

**Operations:**        **GetData()**

**PutData()**

**State machine:**    **No**

**Concurrency:**     **Sequential**

**Persistence:**     **Transitory**

**Class name:**

Adjacency

**Documentation:**

operations based on adjacency.

**Visibility:** Exported

**Cardinality:** n

**Hierarchy:**

**Superclasses:** Proximity

**Public Interface:**

**Operations:** IsAdjacent()

**State machine:** No

**Concurrency:** Sequential

**Persistence:** Transitory

**Class name:**  
Proximity

**Documentation:**  
Operations dealing with proximity.

**Visibility:** Exported  
**Cardinality:** n  
**Hierarchy:**  
  **Superclasses:** Spatial  
**Public Interface:**  
  **Operations:** NearestNeighbor()  
  
**State machine:** No  
**Concurrency:** Sequential  
**Persistence:** Transitory

**Class name:**  
**Containment**

**Documentation:**  
Operations which deal with spatial containment.

**Visibility:** Exported

**Cardinality:** n

**Hierarchy:**

**Superclasses:** Spatial

**Public Interface:**

**Operations:** PointInPolygon()

**State machine:** No

**Concurrency:** Sequential

**Persistence:** Transitory



**Class name:**  
**Aspatial**

**Documentation:**  
Abstract analysis class for data which is aspatial by nature.

<b>Visibility:</b>	Exported
<b>Cardinality:</b>	n
<b>Hierarchy:</b>	
<b>Superclasses:</b>	Spatial
<b>State machine:</b>	No
<b>Concurrency:</b>	Sequential
<b>Persistence:</b>	Transitory

**Class name:**  
**Transformation**

**Documentation:**  
Abstract class for transformations on asptail data.

**Visibility:** Exported  
**Cardinality:** n  
**Hierarchy:**  
    **Superclasses:** Aspatial  
**Public Interface:**  
    **Operations:** CoordXform()  
  
**State machine:** No  
**Concurrency:** Sequential  
**Persistence:** Transitory

**Class name:**  
Neural

**Documentation:**  
Abstract class for neural network models.

**Visibility:** Exported  
**Cardinality:** n  
**Hierarchy:**  
    **Superclasses:** Adjacency Aspatial  
**State machine:** No  
**Concurrency:** Sequential  
**Persistence:** Transitory

**Class name:**

**User**

**Documentation:**

User defined classes - inherited as needed from other classes.

**Visibility:** Exported

**Cardinality:** n

**Hierarchy:**

Superclasses: none

**State machine:** No

**Concurrency:** Sequential

**Persistence:** Persistent

**Class name:**  
**Backprop**

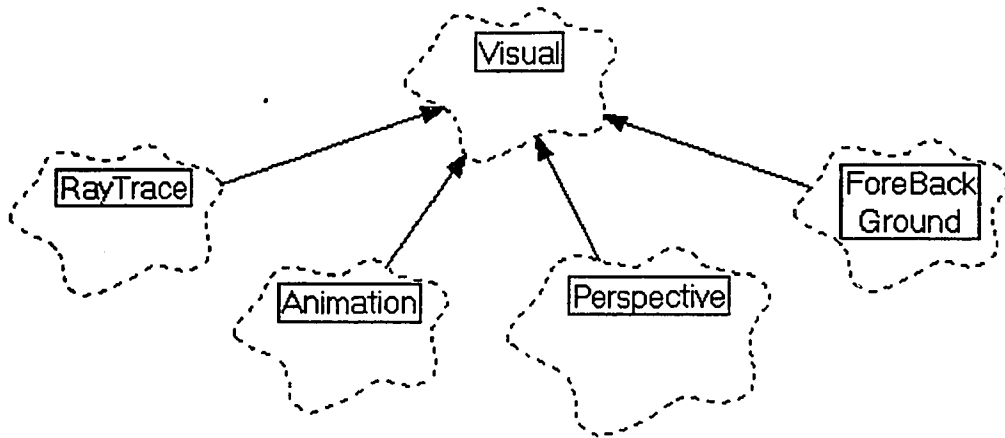
**Documentation:**  
**Backpropagation of error neural network class**

<b>Visibility:</b>	<b>Exported</b>
<b>Cardinality:</b>	<b>n</b>
<b>Hierarchy:</b>	
<b>Superclasses:</b>	<b>Neural</b>
<b>State machine:</b>	<b>No</b>
<b>Concurrency:</b>	<b>Sequential</b>
<b>Persistence:</b>	<b>Transitory</b>

**Class name:**  
Hopfield

**Documentation:**  
Hopfield neural network class

<b>Visibility:</b>	Exported
<b>Cardinality:</b>	n
<b>Hierarchy:</b>	
<b>Superclasses:</b>	Neural
<b>State machine:</b>	No
<b>Concurrency:</b>	Sequential
<b>Persistence:</b>	Transitory



**Class name:**  
**Visual**

**Documentation:**  
Abstract class for display member functions.

**Visibility:** Exported

**Cardinality:** n

**Hierarchy:**

**Superclasses:** none

**Public Interface:**

**Operations:** Display()

**State machine:** No

**Concurrency:** Sequential

**Persistence:** Transitory



**Class name:**  
**RayTrace**

**Documentation:**  
Contains ray-tracing member functions.

**Visibility:** Exported  
**Cardinality:** n  
**Hierarchy:**  
    **Superclasses:** Visual  
**State machine:** No  
**Concurrency:** Sequential  
**Persistence:** Transitory

**Class name:**  
**Animation**

**Documentation:**  
Handles animation routines.

<b>Visibility:</b>	<b>Exported</b>
<b>Cardinality:</b>	<b>n</b>
<b>Hierarchy:</b>	
<b>Superclasses:</b>	<b>Visual</b>
<b>State machine:</b>	<b>No</b>
<b>Concurrency:</b>	<b>Sequential</b>
<b>Persistence:</b>	<b>Transitory</b>

**Class name:**  
**Perspective**

**Documentation:**  
Handles perspective calculations for display.

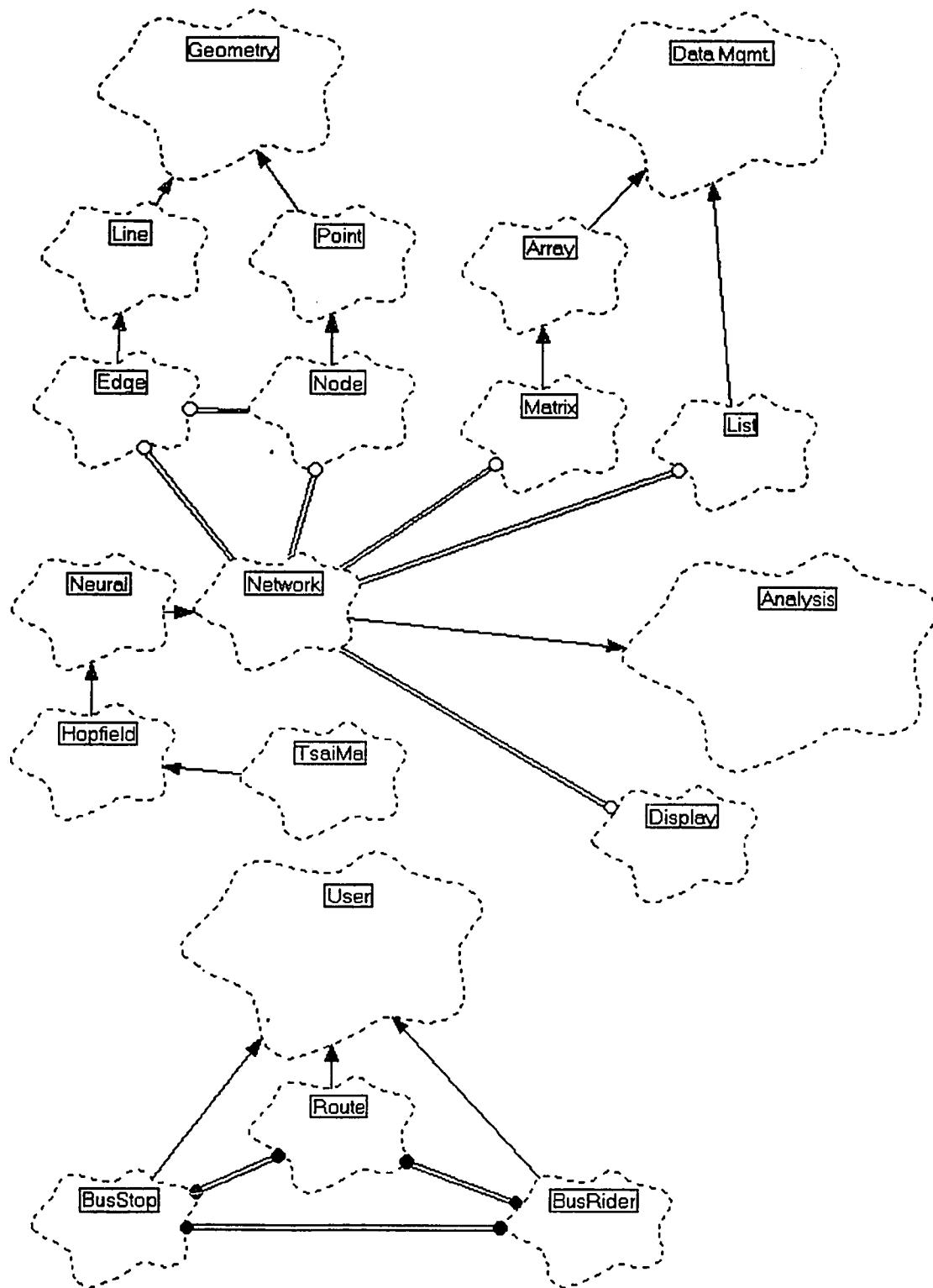
<b>Visibility:</b>	<b>Exported</b>
<b>Cardinality:</b>	<b>n</b>
<b>Hierarchy:</b>	
<b>Superclasses:</b>	<b>Visual</b>
<b>State machine:</b>	<b>No</b>
<b>Concurrency:</b>	<b>Sequential</b>
<b>Persistence:</b>	<b>Transitory</b>

**Class name:**  
**ForeBackGround**

**Documentation:**  
Foreground/background display routines such as hidden  
line removal, etc.

**Visibility:** Exported  
**Cardinality:** n  
**Hierarchy:**  
    **Superclasses:** Visual  
**State machine:** No  
**Concurrency:** Sequential  
**Persistence:** Transitory







**Class name:**

**Geometry**

**Visibility:**           **Exported**

**Cardinality:**           **n**

**Hierarchy:**

**Superclasses:**   **none**

**State machine:**   **No**

**Concurrency:**       **Sequential**

**Persistence:**       **Transitory**



**Class name:**  
**Data Mgmt.**

**Visibility:** Exported  
**Cardinality:** n  
**Hierarchy:**  
    **Superclasses:** none  
**State machine:** No  
**Concurrency:** Sequential  
**Persistence:** Transitory

**Class name:** Line

**Visibility:** Exported

**Cardinality:** n

**Hierarchy:**

**Superclasses:** Geometry

**State machine:** No

**Concurrency:** Sequential

**Persistence:** Transitory

**Class name:**

**Edge**

**Visibility:**       **Exported**

**Cardinality:**       **n**

**Hierarchy:**

**Superclasses:**   **Line**

**Uses for Interface:**

**Node**

**Network**

**State machine:**   **No**

**Concurrency:**     **Sequential**

**Persistence:**     **Transitory**

**Class name:**  
**Point**

**Visibility:** Exported  
**Cardinality:** n  
**Hierarchy:**  
    **Superclasses:** Geometry  
**State machine:** No  
**Concurrency:** Sequential  
**Persistence:** Transitory

**Class name:**

**Array**

**Visibility:** Exported

**Cardinality:** n

**Hierarchy:**

**Superclasses:** Data Mgmt.

**State machine:** No

**Concurrency:** Sequential

**Persistence:** Transitory

**Class name:**

**List**

**Visibility:**           **Exported**

**Cardinality:**           **n**

**Hierarchy:**

**Superclasses:**   **Data Mgmt.**

**Uses for Interface:**

**Network**

**State machine:**   **No**

**Concurrency:**       **Sequential**

**Persistence:**       **Transitory**

**Class name:**

**Matrix**

**Visibility:**           **Exported**

**Cardinality:**           **n**

**Hierarchy:**

**Superclasses:**   **Array**

**Uses for Interface:**

**Network**

**State machine:**   **No**

**Concurrency:**       **Sequential**

**Persistence:**       **Transitory**

**Class name:**

**Analysis**

**Visibility:**           **Exported**

**Cardinality:**           **n**

**Hierarchy:**

**Superclasses:**   **none**

**State machine:**   **No**

**Concurrency:**       **Sequential**

**Persistence:**       **Transitory**



**Class name:**

**Network**

**Visibility:** Exported

**Cardinality:** n

**Hierarchy:**

**Superclasses:** Analysis

**Public Interface:**

**Operations:** FindNeighbors()

Dijkstra

**State machine:** No

**Concurrency:** Sequential

**Persistence:** Transitory

Class name:

Neural

Visibility: Exported

Cardinality: n

Hierarchy:

Superclasses: Network

State machine: No

Concurrency: Sequential

Persistence: Transitory

**Class name:**

**TsaiMa**

**Visibility:** Exported

**Cardinality:** n

**Hierarchy:**

**Superclasses:** Hopfield

**Public Interface:**

**Fields:** energy  
converge\_limit

**Operations:** activate()  
update()  
calc\_energy()

**State machine:** No

**Concurrency:** Sequential

**Persistence:** Transitory

**Class name:**  
**Hopfield**

**Visibility:** Exported  
**Cardinality:** n

**Hierarchy:**

**Superclasses:** Neural

**State machine:** No

**Concurrency:** Sequential

**Persistence:** Transitory

**Class name:**

**Node**

**Visibility:**           Exported

**Cardinality:**           n

**Hierarchy:**

**Superclasses:** Point

**Uses for Interface:**

**Network**

**State machine:** No

**Concurrency:** Sequential

**Persistence:** Transitory

Class name:

Display

Visibility: Exported

Cardinality: n

Hierarchy:

Superclasses: none

Uses for Interface:

Network

State machine: No

Concurrency: Sequential

Persistence: Transitory

**Class name:**  
User

**Visibility:** Exported  
**Cardinality:** n  
**Hierarchy:**  
    **Superclasses:** none  
**State machine:** No  
**Concurrency:** Sequential  
**Persistence:** Transitory

**Class name:**

**BusStop**

**Visibility:**       **Exported**

**Cardinality:**       **n**

**Hierarchy:**

**Superclasses:**   **User**

**Uses for Implementation:**

**BusRider**

**Route**

**State machine:**   **No**

**Concurrency:**     **Sequential**

**Persistence:**     **Transitory**



**Class name:**

**BusRider**

**Visibility:**           **Exported**

**Cardinality:**           **n**

**Hierarchy:**

**Superclasses:**   **User**

**Uses for Implementation:**

**BusStop**

**Route**

**State machine:**   **No**

**Concurrency:**    **Sequential**

**Persistence:**     **Transitory**

**Class name:**

**Route**

**Visibility:** Exported

**Cardinality:** n

**Hierarchy:**

**Superclasses:** User

**Uses for Implementation:**

BusStop

BusRider

**State machine:** No

**Concurrency:** Sequential

**Persistence:** Transitory