1-1-1990

Object Oriented Terrain Databases For Visual Simulators

Brian S. Blau

Object Oriented Terrain Databases for Visual Simulators

by

Brian Scott Blau

B. S., University of Central Florida, 1988

Graduate Committee:
Dr. J. Michael Moshell (Chairman)
Dr. Charles E. Hughes
Dr. Ali Orooji

Project

Submitted in partial fulfillment of the
requirements of the Master's of Science degree in
Computer Science in the Graduate Studies Program
of the College of Arts and Sciences
University of Central Florida
Orlando, Florida

Fall 1990

Object Oriented Terrain Databases for Visual Simulators

by

Brian Scott Blau

B. S., University of Central Florida, 1988

Project

Submitted in partial fulfillment of the
requirements of the Master's of Science degree in
Computer Science in the Graduate Studies Program
of the College of Arts and Sciences
University of Central Florida
Orlando, Florida

Fall 1990

Abstract

This project is intended to develop methodologies and solutions to the problem of representation and utilization of dynamic terrain on a real-time simulator. Until recently, real-time image generation had been focused on flight simulators. With the increased use of ground based simulation, such as armor and infantry, terrain imaging has become vital as well. However, existing simulators are limited in their ability to deal with changing terrain features. This project proposes the use of an object oriented terrain database to handle the problem of dynamic terrain. The main objective of this project is to design and implement an object oriented terrain database manager which will use object oriented data structures to represent the terrain as well as other simulation objects.

Dedication

To Patricia,

Allan, Sally, Jodi and David

with love

Acknowledgements

I would like thank all the people who have made a difference in my life while I was a graduate student. First come professors Dr. Charles E. Hughes and Dr. J. Michael Moshell. Their help, direction and encouragement were always an inspiration to me. Without their guidance and support I would not have accomplished the goals which I had set for myself. Most of all, they gave me social and professional opportunities which I would not have had anywhere but at IST and UCF. I feel honored to have worked with them. I would also like to thank Dr. Ali Orooji participating in my graduate education.

My friends are very important to me. Roommate Kevin Bryden and close friend Michael Gravel have helped me through the best and worst of times. I owe a great deal of friendship to them. I also owe a great deal of thanks to my classmates and friends, Marty Altman, Jennifer Burg, Mahesh Dodani, Richard Dunn-Roberts, Ron Klasky, Curt Lisle, Micheline Provost, Robynn Sebastian and John Turner. I appreciate their help in class work and helping me have fun while in school. I would also like to acknowledge Ernie Smart and all of the supportive people at the Institute For Simulation and Training, Visual Systems Laboratory.

I would also like to thank my family for supporting me throughout my school. Allan, Sally, Jodi, David, Hilda, Carole, Max and Curt have always been there for me when I needed it.

Finally, I would like to thank Trish for putting up with me for the past two years. She has been an inspirational light to whom I give all my love and an enormous amount of gratitude.

Table of Contents

Table of Figures

1 Motivation for Object Oriented Terrain Databases

In recent years, the computer revolution has made possible high fidelity computer simulations. Visual simulators have traditionally employed static terrains over which active objects move. In these environments, actions that should alter the terrain sometimes result in visual changes. The construction of earthworks, erosion and traffic damage are examples of the complex effects which occur in real life and need to be emulated in graphical simulations. Unfortunately, existing systems rarely modify the terrain's internal structure and behavioral characteristics.

Developments in hardware and software are now making it possible to manage and display dynamically changing terrain in real-time. Using an object oriented approach, it is now possible to implement a system which supports dynamic terrain. The project described in this report has developed novel data structures and modeling algorithms for land formations that can be modified during display.

1.1 Motivation

Massive processing power combined with the need for accurate simulations has created the computer simulation industry. This industry focuses its efforts on how humans use computer inputs and outputs to better train for a particular task. Many private and public institutions conduct research to continually improve the training/response scenario. An example of a training simulator is the SIMNET network of tactical tank trainers [Johnson 87]. In this simulation, many separate tank units participate in a single computer networked battle, giving the soldier a feeling of being in a large battle. This device helps the U.S. Army give soldiers effective training in a "semi-real" situation so they might be better prepared for actual battle.

Some computer based simulations are used for training, but there are an equal number of computer applications which are not training situations, but have more social implications. Examples can be seen in Walt Disney World and Universal Studios in Florida. The engineers there have created the Star Wars/Body Wars and Hanna Barbara ride experiences. In these similar rides, passengers sit in a room where they view an animated film. The motions portrayed in the film are felt by passengers as a motion platform moves them around. In some of these cases, computers can generate the film shown to the audience as well as coordinate the control of the motion platform.

This project focuses on how to make computers better represent the real world in a simulation and training environment. Although there have been many advances in computer simulations, there still remain many hard problems to solve. The work described below brings into focus some of the difficulties with today's computer simulations.

1.2 Simulation and Virtual Realities

A **visual simulation** can be defined as any computer simulation in which one of the cues to the user is in some form of computer graphics. In most cases, computer graphics are presented on a high resolution color display. The scenes presented to the user mimic a window to the real world. If one looks out the window of an aircraft cockpit, one sees gross terrain details like mountains, lakes and oceans. The desired effect in a flight simulator is to create a realistic scene when looking out the window of the simulated cockpit. This environment which is located on the other side of the graphics screen is called a **virtual world**, a world which exists only in the mind and imagination of the user and in the computer's memory.

1.2.1 Training Simulators

One of the main customers of training simulators is the United States Department of Defense. Soldiers have traditionally had weapon use and tactical fighting instruction on the battlefield, in which the cost is paid with human lives. With the advent of the computer revolution, the government realized they could use computers to train soldiers in peace time without the loss of life. The military has many uses for training simulators. Much of today's military is very technical. Advanced fighter aircraft contain complicated controls and require years of training to learn. Each aircraft is flown by one or two men and can deliver as much destruction as a battalion of foot soldiers. To use these advanced weapons, a solider must undergo many hours of training. The cost of training a new pilot in actual flight is very high compared to the cost of training on a simulator. Expenses for training in an aircraft come from necessities such as fuel and maintenance. Not only is money a factor, but the pilot's life is in danger every time he flies. But to properly train the pilot, the simulation of the aircraft and the environment must match the experience of actual flight. All of the visual and mechanical cues must be present in the simulated flight.

The simulators which train pilots typically have high resolution graphics in terms of their scene complexity and visual realism. Typically, the fastest graphical simulations will refresh their scene display about 60 cycles/second. The highest fidelity simulators are used to train the most sensitive of tasks such as the operation of fighter aircraft and helicopters. Many trainers do not require high speed graphics systems. In fact, higher fidelity terrain can be achieved with lower update rates. These types of graphical systems are used in ground based simulations, such as tank and driver trainers.

The following are two examples of training simulators which are in use by the military. Each has a description of the trainer and the desired training effect. Also listed are the approximate cost, performance and the manufacturer.

Tank Driver Trainer :

Physical description :
The tank driver sits in an exact copy of the real tank driver's cockpit. The visual system accurately portrays the terrain as seen by a tank driven with the hatch open or closed, and the whole unit is located on a motion platform. A single can be with networked with other units.

Training effect :
Soldier learns how to drive a tank without actually using one.

Cost/Performance :
Visual system : PT2000
Cost: Approximately $1,000,000 for one visual channel

Manufacturer:
General Electric Simulator Division, Daytona, Florida

Tank tactical trainer: SIMNET

Physical description :
The SIMNET network of tanks is used as a mass participation simulation. Many separate tank simulator units are connected together so they can all participate in the same battle.

Training effect :
SIMNET teaches soldiers how to perform in a battlefield situation. It provides a situation in which a solider can learn the tactics of cooperation when in battle.

Cost/Performance :
Visual System: Delta Graphics image generator
Cost : Approximately $250,000 for one tank unit.

Manufacturer:
BBN and Delta Graphics, Boston, Massachusetts

1.2.2 Simulator Components

The complexity of military machines has a direct correlation to the complexity of their simulator counterparts. Simulation units must mimic the actual equipment in both physical structure and virtual environments. Often the training device is built from the plans of the actual machine. For the soldier to perceive the simulation as reality, it is important that computed sensory cues parallel the actual cues in as many ways as possible.

A training device is composed of many complicated mechanical and computational components. A pilot flies an F-16 fighter with a hard stick and throttle. The simulator should also have corresponding hardware components which feel, look and perform the same.

1.2.3 Data Paths

The data flow in a simulator also mimics how a soldier fights in a real battle. First there is some input to the fighting machine, or in the simulator's case, input through discrete switches or analog controls. Next, a weapon is fired and some destruction occurs, the simulator computes the trajectory of the round and it determines if any object has been hit. Finally the soldier sees the result of his action and again inputs some action to his fighting machine. Here the simulator computes the graphical screen and displays it on a monitor. Finally the cycle is complete and input is again introduced to the unit.

User input can come in many forms. There can be triggers, knobs, pedals, switches, steering wheels, throttles and buttons. Once the input has been gathered inside the host simulation computer, the computation system starts. Here all of the inputs are converted to world coordinates so

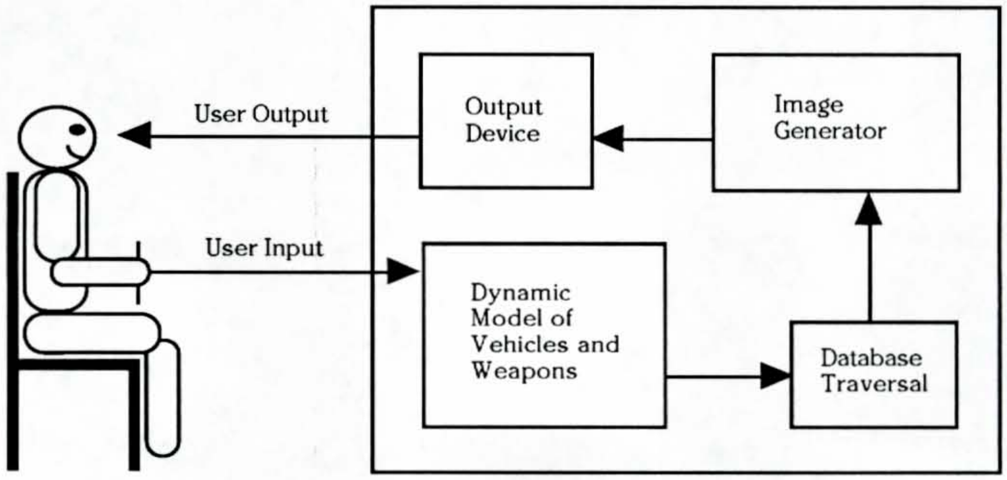

Fig. 1.1 Life cycle of user input to user output

all data is in the same coordinate system. The physical dynamics of the simulation determine the velocity, direction and explosive effects of all objects which are in the simulation. Next, a database is read to determine what environment surrounds the user. This data is then sent to the **image generator** and a picture is displayed. There may also be lights and gauges for the user to read.

1.2.4 Visual Systems

One important output of a simulator is the **visual display**. Here the user is presented with some graphical representation of the virtual world which is located on the other side of screen. This display is generated by a high speed **graphics engine**, commonly called an **image generator**. This functional component takes three dimensional information about a scene and using mathematical transformations, projections and rendering routines, produces a picture display. Inputs to the image generator come from the user's position in the virtual world and a database which contains a model of the world. The output of the graphics engine is through a CRT monitor and video projector.

1.3 Models

A typical simulator database models different aspects of the simulation world. There are models of the terrain, vehicles, houses, trees, forests, roads, lakes, power lines and rivers. Many of these are **dynamic**, meaning they are able to move about the virtual world either by some user input or by automated codes. Other models are **static** in that they never change. A vehicle will use a dynamic model because it needs to move about, and the terrain will use a static model because it will not change.

Models can be described in many ways, depending on their size and complexity. A **polygon** is a planar figure, usually represented by a list of 3 dimensional points. It takes at least 3 points to define a polygon. When drawn on a graphics screen, a polygon will be assigned a color based on its location and orientation. There may be one color for the entire polygon or it may be smooth shaded across its face. Polygons can be used to describe the physical likeness of many models and can either be static or dynamic.

1.4 Fidelity

Simulations can be classified on a spectrum ranging from **low** to **high fidelity**, meaning the components of the trainer are either gross or close approximations of the equipment being simulated. To be able to give the user an accurate representation of the surrounding virtual world, the graphics system must be able to display many different views very quickly. High fidelity simulators typically have display update rates of 60 cycles/second. This means data must be gathered from the database, sent to the image generator and the final picture displayed very quickly. This high update rate is necessary to mimic the human visual system, which is a continuous input system.

Typically, high fidelity systems are high cost and low fidelity systems are low to medium cost. Low cost can range from $100,000 to $300,000 and high cost can range from $500,000 to $2,000,000. Figure 1.2 profiles six actual 1990 visual systems [Johnson 87] [Silicon 90] [ESIG 89] [General 90].

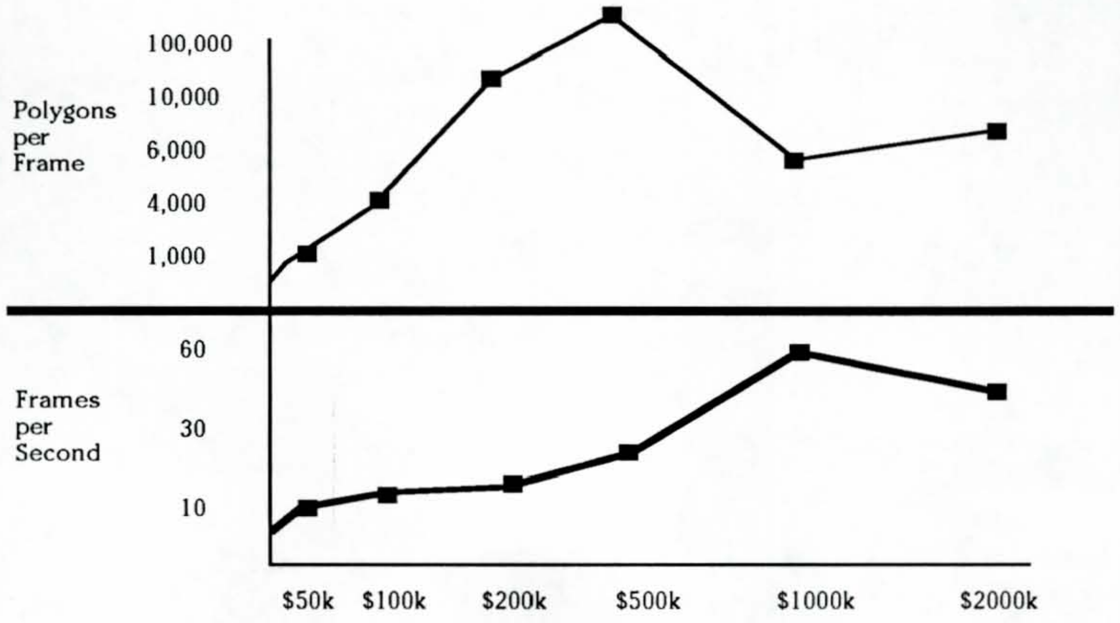

Fig. 1.2 Cost of using an image generator

1.5 Dynamic Terrain

Even though there are many successful image generators on the market, they are all lacking in one respect. Some of the inputs the user is giving to the simulator unit are not being recognized. These may be very subtle inputs, but they still are not handled. For example, if a tank is driving close to a river bank, but not actually in the river, the ground underneath the tank may be soft. When the tank moves it should leave marks in the soft ground. Also, if a tank fires its weapon and the round does not hit another object, it then strikes the ground leaving a crater. A simulation of these events should correspond to their real world counterparts.

If this capability were to exist in simulators, the functions to create craters and tank tracks would be located in the image generator, or some piece of equipment closely connected to it. But the databases and graphics engines in today's simulators do not have this capability.

Both of the events described above modify the underlying terrain. Previously the terrain was named as a static model because it would never change, but it now must be made a dynamic model. We will now call a dynamic model of terrain, **dynamic terrain**.

There are a few possible solutions to the problem of dynamic terrain. When a tank leaves tracks, it may be enough to simply display a black mark where the tank has driven. This is an attempt to leave a permanent mark of the tank's presence. This method is acceptable when only looking at the tracks, but it is not acceptable when trying to navigate them. This effect can be seen in Fig. 3, Tank A. Here textures are left behind the tank as it moves across the terrain.

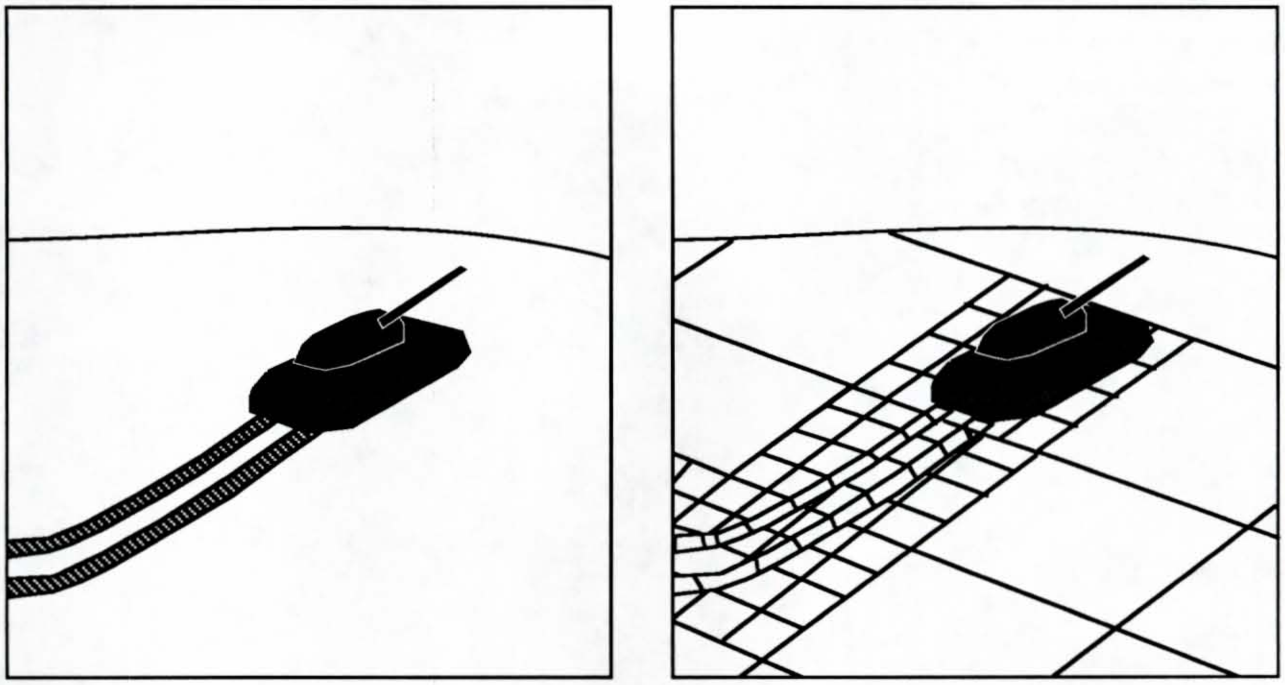

Fig. 1.3 Tank A may look good, but Tank B leaves real tracks

If one tank is in pursuit of another, then the visual and motion cues felt by driving over the tracks might be important. After the passage of a column of tanks, a roadway is often impassible to wheeled vehicles. This effect can be seen in Fig. 1.3, Tank B. As the tank moves across the terrain, changes to the terrain polygons are made automatically. A small depression in the polygon signifies that the ground is lower at these points. The depth of the crater might give the enemy some idea of its size. These visual cues are not part of today's visual systems. The inputs that have been given (e.g. driving over soft ground) are not registered by the simulation computer.

Not only does the dynamic terrain model apply to terrain, but if the model is extended, all forms of databases will be able to share in the dynamic nature of models. As it stands now, the simulation industry has put a restriction on what can and cannot move in a simulation. By making all models of a simulation dynamic, we can better recreate reality.

1.6 Object Oriented Technology

There has been a software evolution which may hold the answer to some of the problems with visual simulators. **Object oriented technology** has the inherent ability to manipulate and manage the complex data structures which are necessary for dynamic terrain.

1.6.1 Objects

Many parts of a simulation can be described as **encapsulated data structures**, or **objects**. A house, tree or tank can be an object. Objects are instantiations of data structures with **actions** and **slot variables**. The slot variables of an object are objects themselves. Sending a **message** to an object requires that object to respond by executing one of its associated actions. Actions can be **passive** or **active** and the only way the outside world has access to that object is through one of the actions. The collection of an object's actions are known as its **protocol**.

Because objects are encapsulated data structures, they are dynamic by nature. By knowing the protocol, objects may send messages to each other without regard to implementation or method of execution. Thus when a message is sent to an object and the action is passive (i.e. the action is simply to assign a value to a slot variable) no message propagation takes place, but if the action is active, then many messages may propagate and affect other objects located in the system. Nesting of objects is possible through the use of slot variables. This is called an **object hierarchy** or **part-whole hierarchy**. One slot variable may contain an object, which has a slot containing another object.

1.6.2 Object Oriented Design for Simulations

A simulator database may be modeled by using objects. Houses, trees, rivers, lakes and terrain are candidates for these encapsulated data structures. A simulation database may easily take advantage of the part-whole hierarchical structure. For example, a house object may have as its parts a door, a kitchen, a fireplace and a room. The kitchen object may in turn have a stove, a sink and a cabinet. The terrain of a simulator database may be modeled the same way. A terrain may be one object, itself comprised of smaller pieces of terrain. Each smaller terrain may contain even smaller terrain objects. A terrain object may even contain houses and trees as parts. This encapsulation of objects gives the terrain a natural hierarchy. Using the object oriented structure to implement the database, both static and dynamic models can be incorporated into the database.

Within an object oriented design, each part of the simulation is independent from the others. Objects may receive messages and act upon them accordingly. Because each object is encapsulated, different parts of the simulation will not know how other parts are implemented. The only communication route is through message sending. This means each object must have a well defined protocol which must be known to all other objects. This is the main restriction on how objects interact, and it is not a drawback, but a feature. Because the implementation is not known to the outside world, there will be no direct modification of the internal variables. This encapsulation leaves the object free to use any implementation without fear of losing reuseability or portability. The state of the object is defined by its internal variables, while modification is done through its standard protocol.

1.7 Objectives and Scope

This project is intended to develop methodologies and solutions to the problem of representation and utilization of dynamic terrain on a real-time simulator. Dynamic terrain is one advancement which strengthens the training effectiveness of a computer simulator. It now seems possible to use object oriented technology as a base in the construction of simulator databases. This project will show that an object oriented paradigm can be used to model simulator databases, which include the inherent ability to dynamically modify the underlying terrain.

The main objective is to design and implement an object oriented terrain database manager which will use object oriented data structures to represent the terrain as well as other simulation objects. The use of data from an existing image generator will show how these designs and algorithms can be applied to today's technology. The database manager was designed and implemented in an object oriented programming language, which includes the use of a permanent object storage system. Also, basic queries and commands have been designed and implemented. Finally, an evaluation of the database performance is given.

The work done in this project will address some of the problems with today's simulators. It does not make any revolutionary recommendations, but it should serve as a guide. The ideas developed here are new and have yet to be implemented on a real-time system. The design and algorithms developed are a basis for implementing dynamic terrain on any image generator, and the analysis done here should lead to work in that area.

2 Object Oriented Technology, Design and Databases

The use of object oriented ideas has recently become popular in the software design arena. The software industry for a long time has argued that analysis and design should focus less on the traditional top-down methodology and concentrate on object oriented analysis and design. These alternative design issues focus on software reuse, modularity and abstraction as a foundation rather than on functional decomposition. This chapter will formally introduce object oriented concepts and discuss object oriented databases, current object oriented technologies and the reasons this methodology was chosen for the design of the terrain database.

2.1 Object Oriented Language Features and Design Methods

The main motivation for object oriented design and programming is software reuse. It has been shown that the functions of a system tend to be more volatile than the data structures [Meyer 88]. Thus programs should be designed with modularity and reuse in mind. The concentration should be on functionality of the data rather than on actions.

The first step in designing an object oriented system is to determine which objects exist. This can be done by looking at the data which must be manipulated. A well organized software system can be viewed as an operational model of some aspect of the world. A simulation is an excellent example of an operational model of the world. If the design of this model is well understood, then the world can be viewed as a system of objects. The world being modeled is made up of many different objects which can be houses, trees, tanks and terrain. Here we will find a natural progression from the design of a system to its implementation. In gemeral, the objects are obvious and easy to find [Coad 90].

2.1.1 Objects, Classes and Protocol

An **object** is an encapsulated structure of both **data** and **operations**. It is a single entity which exists independently of other objects. The data are often called **instance variables** and operations are called **methods**. This structure provides modularity and information hiding. It is important to see how objects differ from conventional data abstractions. There are many features of objects that distinguish these two similar concepts. The differences can be seen through the use of **inheritance** and **polymorphism** [Wegner 87][Johnson 88] (see 2.1.2).

An object is a specific instance of some **class**. A class is a template from which many objects can be created. The following is an example of a class definition.

```
Class Tank
        Instance Variables
              gun: an instance of class Gun
              turret: an instance of class Turret
              ammo: an instance of class Ammo
        Methods
              fireWeapon
```

All objects which were created from the same class have the same structure and will have similar behavior [Wegner 87]. Classes may be placed in a hierarchy in which a subclass is a child of a class and a superclass is a parent. Subclasses will **inherit** from their parent classes and superclasses are available to inherit from. All attributes and actions associated with a parent will be accessible to subclasses. Any object created from some subclass will respond to all inherited actions without knowledge of who has defined that action. It is interesting to note that in many object oriented language implementations, a class is simply another object which can respond to message. Thus the class Object (see 2.1.2) is an instance of the class MetaClass. There is a circularity implied here because every class cannot be an instance of another class, meaning the hierarchy would never stop [Goldberg 89].

There are two separate hierarchies that are distinguishable. A **class hierarchy** is defined as a collection of classes in which some are parents and some are children, and a class may be both a parent and child. There exists one class which is the root of the hierarchy in which all classes inherit from. If a class has only one direct parent, then it is said to have **single inheritance**. If it has multiple parents, then it is said to have **multiple inheritance**. Multiple inheritance is conceptually difficult since there are so many ways in which inherited classes may be combined [Wegner 90]. Each application of multiple inheritance implements the conflict resolution in different ways. This problem leads to unportable software systems.

Objects have attributes and actions as part of their foundation. The attributes of an object may be objects themselves. An **object hierarchy**, also called a **part-whole hierarchy**, is a tree of objects in which one object contains other objects. An object may belong to many other objects.

The set of messages that can be sent to an object is defined as its **protocol**. Protocols are standard interfaces between objects, and the only way to communicate information to an object is through its protocol.

Here is a class hierarchy of a tank. It has parts described above with some additional components. Also shown is an object hierarchy. It represents an instance of one tank.

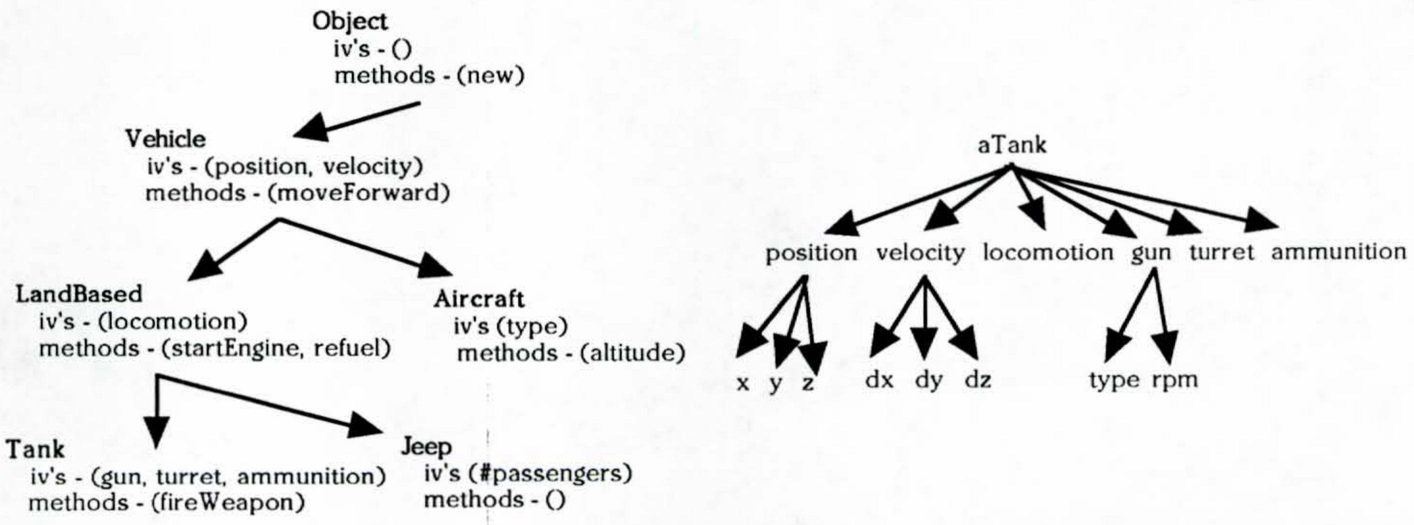

Fig. 2.1 The class hierarchy of a **Vehicle is given on the left.** **Tank** inherits all instance variables and methods from its superclasses. An object hierarchy of one tankis given on the right. The instance variables **position**, **velocity** and **gun** themselves have attributes.

2.1.2 Polymorphism, Inheritance, Messages and Methods

Sending a **message** to an object causes it to perform some particular action which is determined by the name of the message. These are not operating system signals, but late bound functions calls. When a message is sent by one object (**sender**), another object (**receiver**) of the message must determine which of its actions (**methods**) to execute [Johnson 88]. For example, to create a new object in Smalltalk-80, we may send the following message :

```
aNewTank <- Tank new
```

Although the method **new** is not located in the definition of Tank given above, it is inherited from the class **Object** from which all objects must inherit [Goldberg 89]. When the Terrain receives a message, it allocates memory for a new object and assigns it to the variable aNewObject.

Polymorphism is defined as "the ability to take several different forms." In an object oriented language, this describes the ability to refer at run-time to different instances of various classes [Meyer 88]. For example, if there exists a list of objects which are numbers, each object in the list should respond to the message + **(plus)**. If this message was sent to each object in the list, the result would be the total sum. Each object should respond to the message without regard to its class. The objects in the list may be Integers, Floats or Complex numbers. This is similar to operator overloading where the action is bound at compile time. An object oriented system will consider the + **(plus)** a method and bind it late.

This idea of blind message sending is a plus to many systems. It relieves the programmer of knowing the object's type at code time because the message is bound at run time. If this feature was not available, then explicit type checking must be added to every function. This is a dangerous practice because any added modification may cause many changes in the case checking. This leads to software that is difficult to maintain and reuse [Johnson 88].

Inheritance, another feature of object oriented programming, has its own advantages. Inheritance promotes code reuse; code that is common to many classes can be placed in a superclass. This relieves a class from implementing a complete functional system. One additional advantage which inheritance offers is standard protocols [Johnson 88]. Many non-polymorphic languages do not allow different procedures to have the same name. In an object oriented language, this is encouraged, and is the real power of polymorphism.

2.1.3 Abstract and Concrete Classes

Standard protocols are often represented by **abstract classes**. An abstract class never has instances, only subclasses. The root classes of a hierarchy are usually abstract while the leaf nodes are never abstract. They almost never define new instance variables, except for implementation details. They do define methods at different levels, some of which may be part of the standard protocol. Any method which has been defined and has no action associated leaves the implementation details to user defined subclasses. These undefined methods are placed in an abstract class so that the uniform protocol is not broken. Abstract hierarchies are created to be used by others. They should be portable and their implementation is hidden from any subclasses.

A subclass which is not abstract is **concrete**. In general, concrete classes are always the leaf nodes of a class hierarchy and all objects are instances of some concrete class. Since abstract classes do not provide a data representation, concrete classes are free to choose any implementation that is appropriate without fear of superclass conflicts [Wegner 87].

2.2 Object Oriented Databases

An important aspect of using a object oriented design in a terrain database is to be able to permanently store the data. Since this project is concerned with the design of the terrain class structure and the specialized queries which pertain to spatial relationships, it necessary to take a detailed look at object oriented databases (OODB). Most important is the design of the OODB, the storage technique and its query processing capabilities.

An **object oriented database** is similar to a traditional relational, networked or hierarchical database in respect to its storage capabilities and query processing. An object oriented database is another model of data storage which has its roots in object oriented programming. [Kim 90] has given a good definition of object oriented databases which will be summarized in the next few paragraphs.

An object oriented database has a direct relationship to an implementation of an object oriented system. A database is defined as an implementation of some data model, regardless of its base structure. An object oriented model is a logical organization of objects, constraints on them and the defined relationships among the objects. An OODB is the system which directly supports that data model.

An object oriented data model captures the semantics of an object oriented system and is based on object oriented concepts. It captures the relationships between objects, which is the true base of an object oriented system. One of these relationships is aggregation. The **aggregation relationship** in an object oriented data model is simply that a class consists of a set of attributes. The domain of an attribute may be any class. This is one of the main departures from a relational database model where the domain of an attribute is restricted to be of a primitive type. For example, the class Tank has the attribute gun which is an instance of class Gun. Because the domain of an attribute is arbitrary, it leads to a hierarchical structure. Using the Tank as an example again, the gun of the tank also has attributes, such as trigger, barrel and safteyPin.

The fact that the domain of an attribute is arbitrary gives rise to the nested definition of a class. In this hierarchical structure, there is one root class which has attributes that may be other classes or primitives. This model is very similar to the nested structures of a nested relational and hierarchical database. But in an OODB, the class structure forms a directed graph rather than a hierarchy.

The OODB model also includes the **generalization relationship** which is not present in the relational or hierarchical models. This relationship describes the class/subclass structure where a class is a generalization of a subclass. The main feature to note is that an OODB has both the aggregation and generalization relationships, where a class has a role in the class hierarchy and the class graph.

The definition above describes the basic OODB model, but more discussion is required to fully explain an implementation. An OODB must have disk storage and optimization techniques which each database must follow. Most OODB's follow some traditional sector clustering technique where objects of the same class reside close to each other on disk. Other clustering techniques bring together objects of the same user or objects which recursively reference each other.

The other major component of an OODB is the **query processor**. Queries in a traditional database environment are generated by user requests or by automatic response to a user action. In either case there must be a mechanism that can parse the queries and efficiently evaluate the requested information. It is interesting to note that early OODB queries were processed in a manner similar to relational queries. Here a query was directed to a single target class. Recently there has been development in the accommodation of queries to more than one target class [Banergee 88].

Implementations of OODB's have recently become available and all support long term data persistence, transaction control and query processing. Since this technology is new, there has not yet been a standard convention on the interface or model of an OODB. Because of this turmoil, it is difficult to see any trends or future directions.

2.3 Current State-of-the-Art in OODB's

The following is a descriptive list of OODB's that are available as independent database products. There are some OODB's which exist as part of larger systems. These will not be presented here. There are at least

Product Name	Vendor
G-Base	Graphael
GemStone	Servio
0_2	Altair
IRIS	Hewlett-Packard
ITASCA	ITASCA Systems
Object/1	MDBS
Objectivity/DB	Objectivity
OBJECT-STORE	Object Design
OBJECT-BASE	Object Sciences
ORION II	ARTEMIS
Ontos	Onthologic
Persisit	Juniper Software
Statice	Symbolics
Versant	Versant
Vision	Inovative Systems

fifteen products which claim they are OODB systems. The list above shows the products and their vendors [Reit 90] [Versant 90] [Atwood 90] [Deux 89] [ITASCA 90[Objectivity 90] [Banerjee 88]. Three of these products, GemStone, O_2 and ITASCA, are described below. This is to give the reader a sense of the OODB market place in the fall of 1990.

2.3.1 GemStone

GemStone by Servio Logic [GemStone 90] is one of the more popular OODB management systems. It is available on many popular workstations including SUN, IBM PC and Macintosh. Full support is provided for an object oriented data model with objects, classes, instances and messages. It supports inherited class and object hierarchies and comes with a programming language called OPAL. Gemstone also supports an identity feature where every individual object has an object oriented pointer (OOP). This is how speed is achieved in the database manager. When an object is referenced, its OOP is passed around rather than the complete object.

GemStone is a multi-user/multi-application product and runs under a client/server architecture. It can support numerous consecutive users on multiple workstation while coordinating all of the control, transactions and security.

The server of GemStone consists of three components: the **database**, the **resource monitor** and multiple **data servers**. The database is a single logical file on the host system. Internally, the database is a series of logical pages where objects are written. The function of the resource monitor is to synchronize resource allocation and maintain data integrity. The data servers have a large responsibility in GemStone. They map the logical page object storage into the object oriented data model that is presented to the user. They also process all OPAL code that is either in the database or is supplied from the user.

Two other important parts of GemStone are the **user application** and the **platform specific environment**. The user application will interface to GemStone through the environment and a programming language called OPAL. OPAL is very similar to in syntax and semantics Smalltalk-80. They are very similar in syntax and semantics. To make GemStone compatible with Smalltalk-80, the vendors provide a detailed class hierarchy which contains many Smalltalk-80 like classes. A GemStone Class Browser, Workspace and Debugger and user interface are included in their Smalltalk-80 support. Also supplied is a

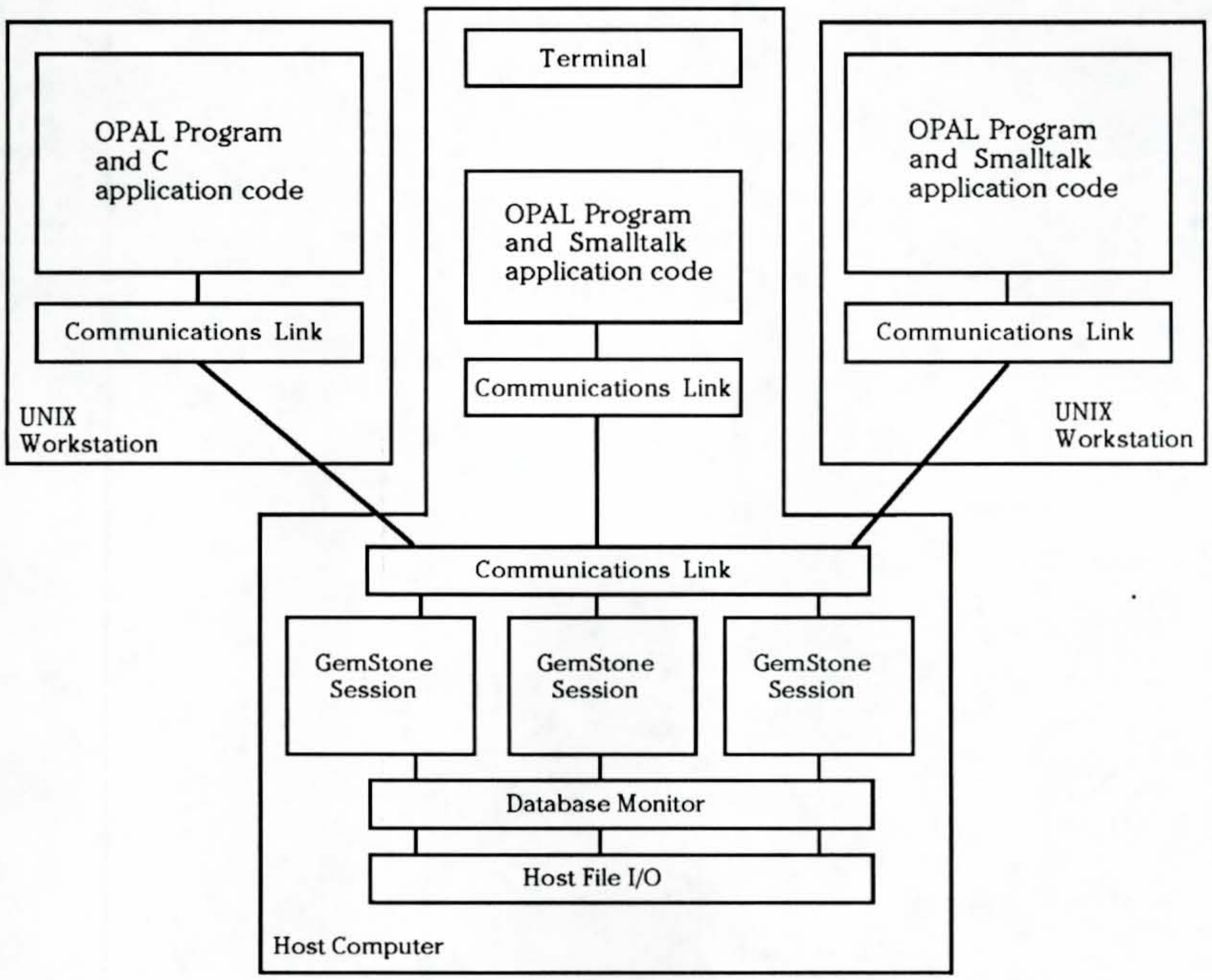

Fig. 2.2 GemStone system architecture

C interface where function calls are used to access database objects. Figure 2.2 is a pictorial view of the GemStone architecture which shows how Smalltalk-80, UNIX and C can all be employed as user platforms to interface with GemStone.

2.3.2 O_2

O_2 [Deux 89] is another OODB management system that uses a programming language, graphical interface and application development toolbox. One of the main selling points of O_2 is the integration of the database and application language. GemStone supports this feature with its Smalltalk-80 like language called OPAL, but ITASCA supports access from traditional programming languages. It supports a full object oriented data model where users have access to all of the nice tools they are accustomed to using.

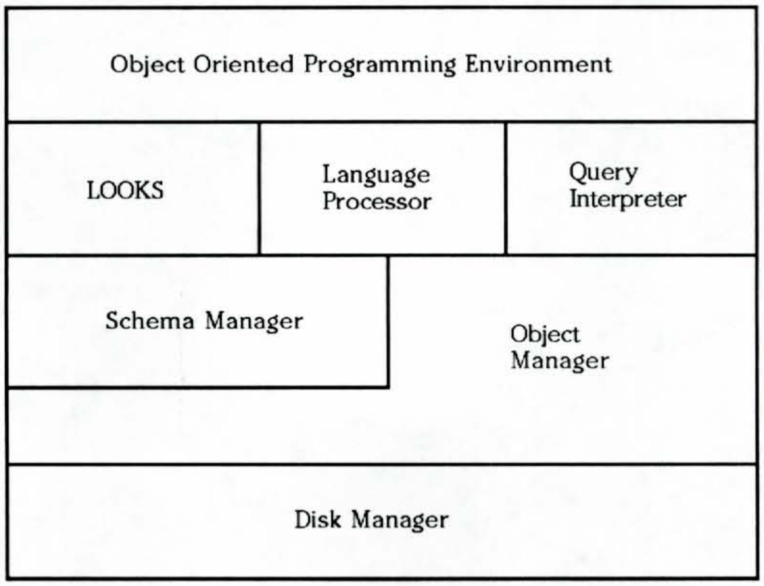

Fig. 2.3 Functional Architecture of O2

O2 consists of eight functional modules. The **Disk Manager** handles low level I/O and base data manipulation. The **Object Manager** has the responsibility of taking the abstract data model and mapping it to its disk representation. The **Language Processor** compiles and stores methods which are then used by the **Schema Manager**. Additionally the **Query Interpreter** takes input and, using the Object and Schema Manager, processes queries. There are also two modules to manage the graphical interface and programming environment. Figure 2.3 shows the relationships of these modules.

The data model of O2 is similar to other OODB's. It maintains a class hierarchy where each object is an instance of a class. Classes describe common behavior among a set of objects, and have attributes and actions associated with them. All objects are manipulated with the methods defined in a class. Methods can be private, in which case only other methods in that class can use them, or may be public and open to all classes. O2 also has user defined types. Types are similar to objects, but they have no action associated with them and they cannot be instanced.

Instead they appear as components of a system and are not part of the object hierarchy.

O2 also supports subtyping and inheritance. Subtyping defines a relationship between two types. A type is a subtype of another if every instance of it is also an instance of its supertype. Inheritance in O2 has the same meaning as defined above (section 2.1.2) but it also supports multiple inheritance. O2 is supplied with two programming languages which are supersets of C and BASIC.

2.3.3 ITASCA

ITASCA Distributed OODB [ITASCA 90] is an advanced distributed database manager. It supports all of the features common to the other OODB mentioned above. There is one aspect of ITASCA which should be noted. It uses a distributed architecture with private and shared objects across UNIX platforms. This architecture allows any number of databases to exist on multiple workstations. The data manager stores a copy of the entire schema on all sites to improve performance. This is a departure from the confining server/client relationship in GemStone and O2. This advantage prevents a single point failure because all schema information is stored on all nodes. When a node is off-line, only its data becomes

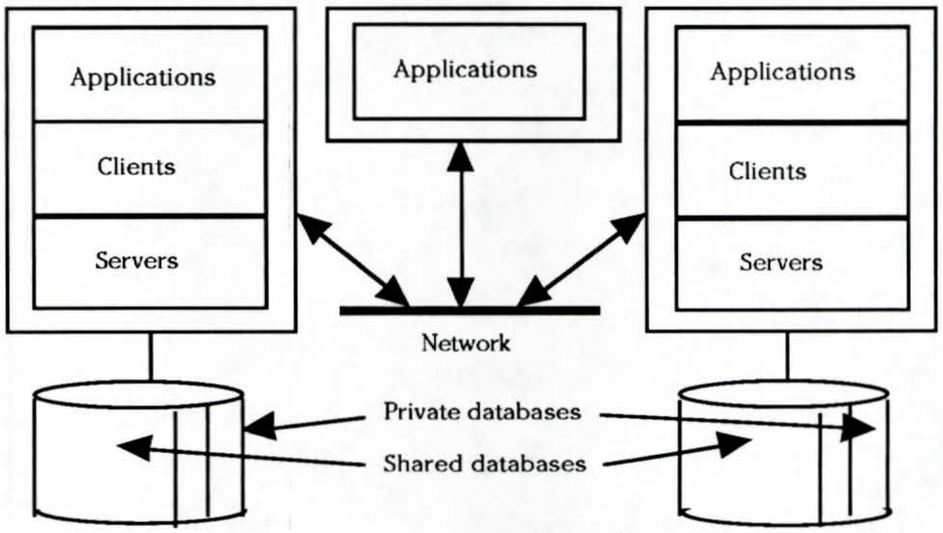

Fig. 2.4 ITASCA Distributed architecture

unavailable, not the class structure. Therefore, editing of data and class reorganization may continue while a node is off-line. Figure 2.4 shows the architecture of the ITASCA system. The clients maintain page and object buffers. Here objects are mapped from object space into disk partitions. The servers maintain object buffers for the local memory, similar to a cache system. A single client can access multiple servers.

2.4 Why an OOBD Methodology Was Chosen For This Project

In general, the OOBD products that are available have been developed and are targeted for large scale data manipulation mostly for use in the business environment. They all contain some programming language and a graphical interface in which to build applications and manipulate objects. Each has a particular object oriented data model which supports classes, objects, class and object hierarchies, and inheritance. Some even have distributed architectures which aid in data management.

The design of this project is object oriented by nature and requires the permanent storage offered by an OODB system. The terrain database is a part of a visual system and all components need access to its data. One method is to employ an OODB to manage all transactions between simulator components. This will relieve the application program from managing any of its data. It will all reside with the underlying support system [Hughes 90].

A terrain database is similar to any other database in that they all must have some efficient permanent storage medium. In this case we have an object oriented design and would like to have persistent objects which may be accessed on multiple platforms. The following diagram is a conceptual picture of a network of simulators and their relationship to the OODB. The networked model shown is a client/server relationship, but because the applications do not rely on data manipulation, it may be reconfigured to be a distributed model. Objects and classes will permanently reside in the database server. Each simulator unit will access objects remotely and will let the database server manage all concurrency and transaction control as well as security. The interface from the data server to the simulator unit is provided through a database interface. This interface is transparent to the application program. Each application views

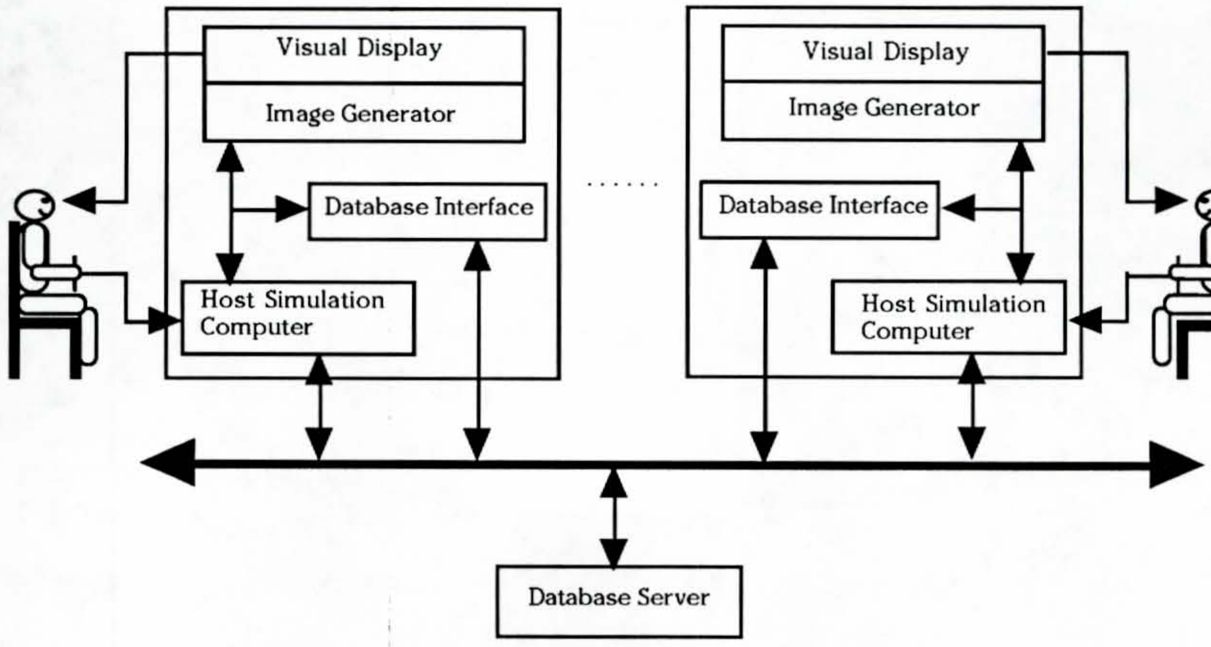

Fig. 2.5 Client/server object oriented terrain database

all of the objects as local entities, members of their own local memory, but in reality, all objects reside permanently on the database server. Any change to an object will be reflected in the main database as well as in any local unit.

3 Spatial Data Management and Terrain Databases

This chapter is a detailed review of two technologies which when combined can help make an object oriented terrain database a reality. The development of spatial data management systems is an area of very active research. The goal is to combine data management and query techniques with three dimensional geometry. This results in a system which accepts queries specifically tailored toward spatial data management. The other technology has been presented in the first chapter, which is simulator database management. This chapter will take a detailed look at different terrain database systems. In chapter 7 we will use these technologies to look at possible future work where a combination of management schemes will produce a faster and smarter terrain database.

3.1 Spatial Databases

Conventional database management has proven insufficient to model certain types of complex data management schemes which involve spatial information [Egenhofer 88a]. Such systems include VLSI design tools, CAD/CAM packages, geographic and land information systems, and military applications such as autonomous vehicle control and battlefield data fusion [Egenhofer 88a, Antony 88]. The use of object oriented technology has provided a way to efficiently implement and model spatial systems without losing processing or design power.

Real world objects and their relevant operations are inherently spatially oriented and are modeled naturally in an object oriented system [Egenhofer 88b]. They require operations that use three dimensional properties of points, lines and areas. For example, the traditional **intersects** operation can be applied to a three dimensional object, where the response will be true if the two objects share at least one point in space. Modeling real world objects is one of the motivating factors behind the move from specialized spatial processors to higher level data management systems.

3.1.1 Deficiencies of Conventional Databases and Data Structures

Most relational or hierarchical database systems do not meet the demands of spatial data management. The composition of spatial data is too complex for a traditional database. The level of detail required for one complex object may exceed the limits of the management system. This can be seen when looking at the performance of some geographical information systems. An interactive system requires high speed manipulation of data without sacrificing performance due to the size of data sets.

In the description of the PANDA [Egenhofer 88b] database management system, Egenhofer cites examples of deficiencies of currently available systems. Performance of engineering systems was shown by Wilkins, Harder and Maier to suffer when large amounts of data were used. Complex objects such as those in chemical systems or VLSI circuits cannot be supported in a non-hierarchical structure. Lorie and Batory stress that spatial objects should not be artificially decomposed into smaller parts.

The relational model does not support the data types necessary to manage spatial data. In most relational systems, the base types are strings, integers and floats which are in a table format. Spatial data is naturally defined by nested structures in some hierarchy. The relational model is not suitable for recursive traversing of the spatial data structure.

There has been work done in this area to help define the needs of a spatial data system. Egenhoffer and Frank have developed the PANDA system, a spatial object oriented database. Manola, Orenstein and Dayal have developed an object oriented geographic information system called PROBE. Antony has been working in the area and his work not only includes spatial data management, but also includes terrain management, tactical data fusion and autonomous vehicle control. These topics will be discussed further in the following sections.

3.1.2 PANDA DBMS

PANDA (Pascal Network Database Management System) [Egenhofer 88a, 88b] is an object oriented spatial data management system. It incorporates all aspects of an object oriented design with special operators to manipulate three dimensional data. The data structures are present for efficient storage and access of objects. The following is a diagram of PANDA's architecture :

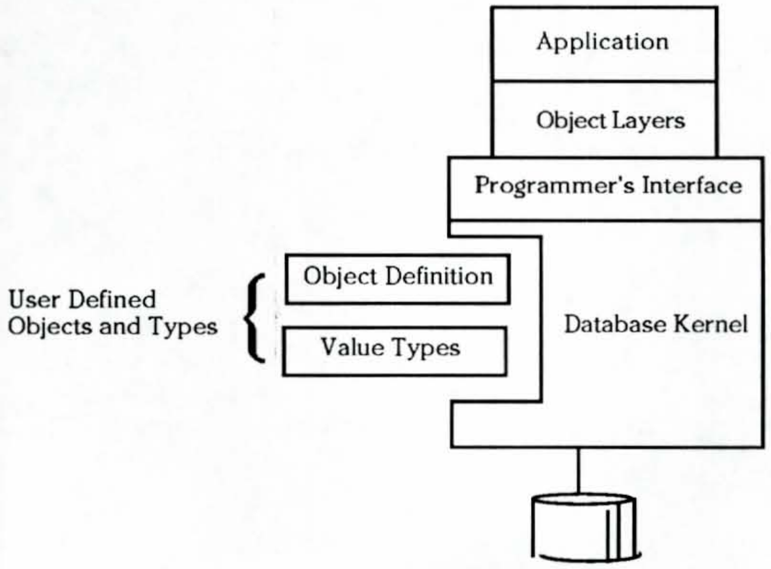

Fig. 3.1 Architecture of PANDA

The **database kernel** manages the storage and retrieval of objects from a permanent storage device. It also provides transaction management as well as logical structure access. The kernel is precompiled but extensible. Users can extend the kernel by using a collection of object oriented operations. These are called as functions and procedures in PASCAL and are defined as the most general actions available to objects. This notion is similar to the Object class in Smalltalk-80.

The definition of an object in PANDA is similar in concept to the definition given above. All objects are instances of a class, and class definitions can be nested into a hierarchy. Objects have some predefined attributes which give them their spatial data representations. The spatial class types are :

Node, Edge, Face, Bounds, SpatialObject

A class can be an instance of one of the above types or can be user defined. Specific object-operations are needed for each object type that is supported. Because PANDA claims to be extensible, the basic object-operations are not defined by the system and are left for the application programmer. For example, a user defined class must supply the operations for hashing because the kernel uses hashing for efficient storage and retrieval. If the class does not define a particular required operation, it may rely on the inherited operation if it exists.

Although PANDA seems like a system which is suited well for operations necessary to manage a terrain database, it lacks many aspects of an object oriented system. One major drawback is that PANDA is compiled PASCAL. This limits the functionality which is possible in systems like Smalltalk-80. The programmer can create objects at runtime, but not classes. Although this aspect of database management may not seem appropriate, it becomes useful when an automatic data manager determines that some attributes and actions can be collected and generalized, automatically reconfiguring the class hierarchy [Hendley 90].

One final comment is on the built-in use of a spatial data management system. In all of the documentation on PANDA they do not describe any implementation details of spatial structure manipulation except for some default class definitions given above. They do not describe how to use this aspect of their system. It is assumed that there are actions which are provided to manipulate spatial data. The nodes, edges, faces and boundaries are used as an efficient disk storage technique and are not necessarily used for geographical queries.

3.1.3 PROBE Research Project

The PROBE research project by Orenstein and Manola [Orenstein 86, 88] has produced a mature spatial object oriented database management system. It was mainly built for databases which require non-traditional query and storage techniques. PROBE is extensible and supports a full object oriented data model. One of the main features is its ability to efficiently store, retrieve and process spatial information.

The PROBE data model has two basic types: **entities** and **functions**. An entity is a data structure which denotes some particular object, which may represent a real world object or an idea. Entities with similar characteristics are grouped together into entity types. In addition to these entity types, functions differentiate entities by defining their properties as relationships between entities and the operations on entities. These two components of the PROBE data model are similar to the definition given above of an object. The type ENTITY is similar to the class Object and is at the top of the generalization hierarchy. All generic operations on objects are defined in the type ENTITY.

The architecture of PROBE comes from a combination of object oriented concepts and database management techniques. It has two major components, the **kernel** and a **collection of classes**. There is a unique division of labor between these two components. The main question is, what part of the query should be processed in the kernel and what part in the class? One possibility is to process all queries in the kernel. This requires that spatial information be hard coded into the kernel and forces the system to be unextensible. Another possibility is to process all of the queries in the classes. In this case, classes must know intimate details of the base data management system. Again this is unacceptable.

A combination of these two techniques is the current PROBE architecture. It consists of limiting which parts of the query are processed in the kernel. Only the most general spatial types are supported in the kernel, ENTITY and POINT_SET. An instance of POINT_SET contains a set of points which represent the space occupied by an object. An example of this division of labor can be seen in the following PROBE query :

```
for each x in S
    for each y in S
        if overlap (x, y) then output (x, y)
```

The database kernel implements the scan of S while the object is responsible for determining if the two objects overlap.

The most interesting aspect of this type of processing is the implementation of the database spatial kernel. Figure 3.2 shows the architecture of PROBE.

The representation that is used by the **geometry filter** is a grid structure. The cells of the grid which contain part of the point-set instance of an object is the set of candidate areas. This is noted as a conservative approximation of the object. When a query is processed, this approximation is found inside the kernel and then passed to the object for further refinement. This is the breakdown of responsibilities between the kernel classes and application classes.

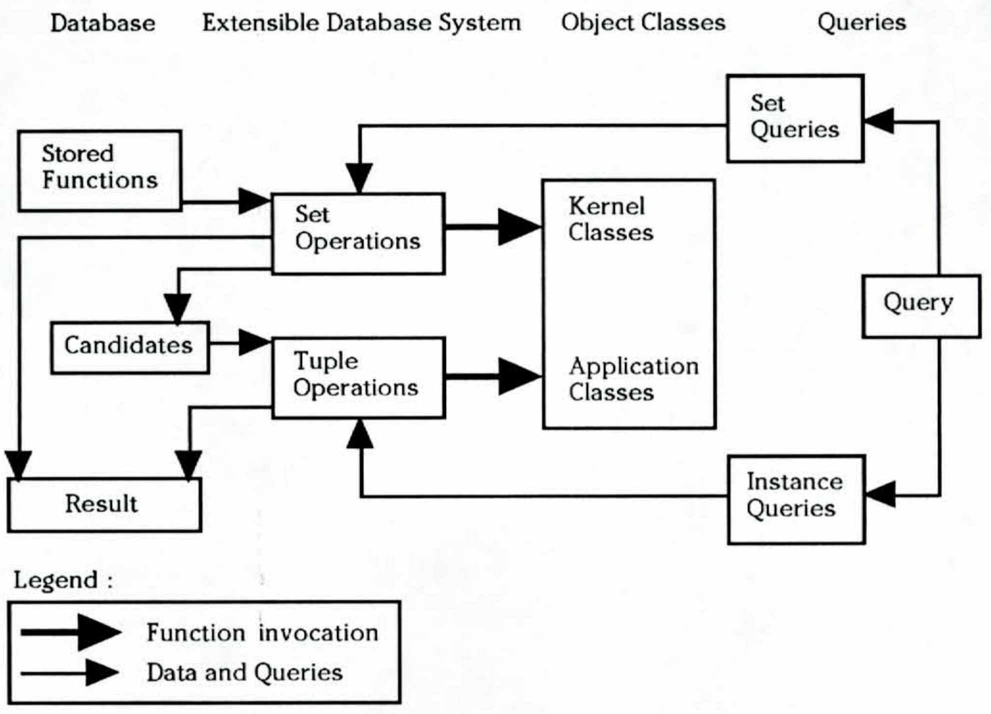

Fig 3.2 Architecture of the PROBE database system

For the geometry filter to effectively process the spatial data, an encoding of the grid is used. This code, called the **z-value**, is similar to a quad tree representation [Gargantini 82] where locations can be described by a unique number. Because this encoding is consistent across grids, the z-value for the location of one object can be compared to another. The grid can be partitioned recursively, making each code more accurate.

With the encoding and the processing responsibility in place, the PROBE system efficiently processes spatial data requests. Operations such as spatial join, POINT_SET_union, POINT_SET_intersect, and POINT_SET_difference are explained fully. Their implementations and analysis are given.

PROBE has many of the features which are needed for the processing of terrain data. It has an object oriented data model which supports classes, objects, inheritance and polymorphism. It also supports a spatial data representation which is needed in any system which has terrain processing. The main reason for examining PROBE is to get ideas on the components needed for the object oriented terrain database. Because the terrain database is hierarchical, it may be possible to incorporate the spatial processing power into the natural hierarchy of the terrain.

3.1.4 Antony's OODB Management System

Antony [Antony 88, 90] has been looking at alternative methods for representing terrain, autonomous vehicle control, target recognition and battlefield data fusion. His proposed database management system is a fully integrated hierarchical, object oriented spatial data management system. Although this system has never been implemented, it represents the latest research ideas in this area and has close ties to object oriented terrain databases.

Because this system is in the proposal stage, it is difficult to determine if an implementation will meet the expectations of the designers. They have one of the most unique ways of looking at the problem of representing spatial data. Their main goal is to have an efficient representation of three dimensional data and at the same time incorporate object oriented technology. The combination of the two is natural when comparing hierarchical representations. It was better said by Antony :

> "If both the object-oriented and spatial-oriented database representations are hierarchical, significant reductions in search space size and subsequent improvements in efficiency of both logical and global problem solving can be achieved. Such a structure ... provides a representation that supports reasoning that is metaphorically similar to that employed by human problem solvers."

The framework of the database consists of four frame based forms which are the **semantic object**, **pyramid object**, **vector object** and **region object**. The semantic object has an attribute-slot form with associated pointers and procedural constructs. The pyramid form summarizes the point, line and region features into a the bit-coded vector. The vector object is a list of continuous line features. The region object is a minimal quadtree representation of a two dimensional region.

The organization of the database allows complex object oriented queries. The query processing will be in two stages, the first links all point, line and area features together into a fully registered low resolution pyramid. Here is where substantial reduction is search space can be found using the hierarchical structure. The second stage of processing refines the representation to the exact precision needed.

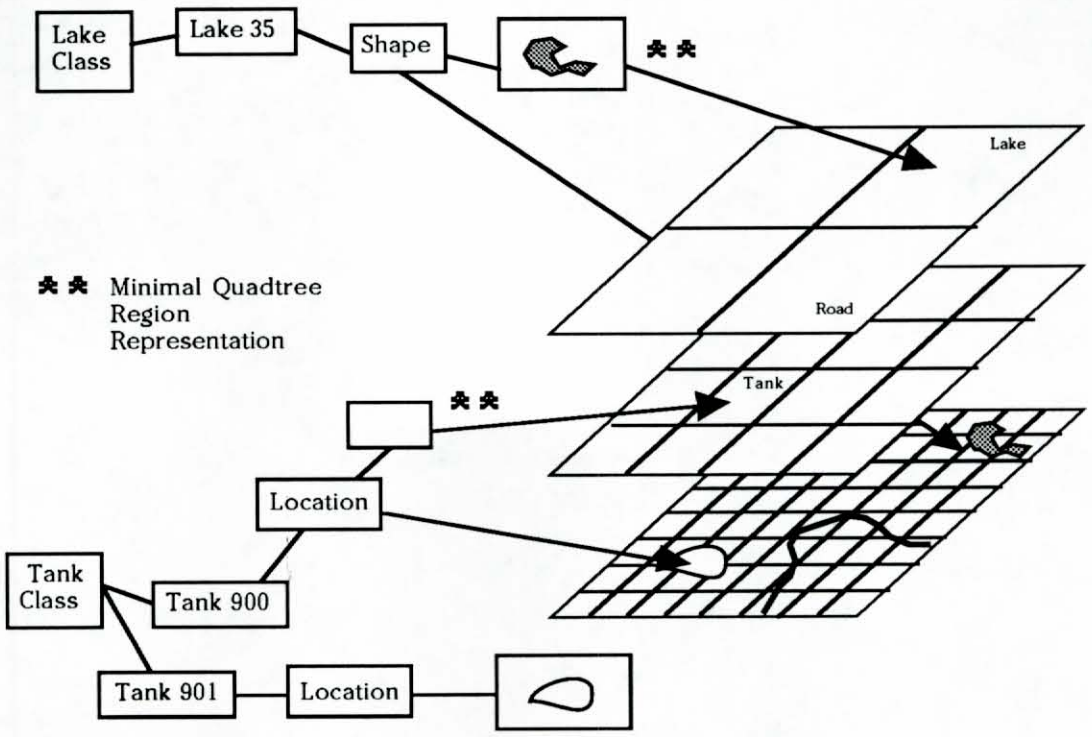

Fig. 3.3 Composite view of Antony's spatial database management system

Figure 3.3 shows an example of the proposed data management system. There are two classes, Tank and Lake, from which instances are created. These objects are the semantic object representations. The pyramid representation can be seen in the quad-tree structure which successively refines the resolution of the terrain. The vector representation is used to encode the road, and is not shown graphically above. The region objects are given as the lake and area.

3.2 Geographic Information Systems

Geographic Information Systems (GIS) are used in land use and land planning. They are used by city planners and engineering firms to properly determine the location of new projects and public utilities. GIS typically use specialized processors or workstations as platforms.

These systems have become popular recently because of their low cost and availability. The reason that the GIS technology is discussed here is its relationship to database queries and spatial data management. These systems answer queries like the following :

```
Areas
where population < density 5 people/square acre
where location < 2 miles from any elementary school
where location < 5 miles from water treatment plant
```

This query would answer all land parcels which meet the criteria. It would show a land developer possible places to build new homes. To process these types of queries, the GIS must know spatial information as well as attributes of an area.

Some of the currently available GIS systems have powerful capabilities. [GRASS 86] is a public domain GIS which is available through CERL at the University of Illinois. Although it is not the most powerful GIS system, it has the capability of processing standard Defense Mapping Agency (DMA) data as well as most query functions necessary to support a small GIS shop. The InterGraph Company [TIGRIS 90] [Karimi 90] manufactures a computer line which is specifically built for GIS systems. They have successfully combined hardware database optimizations with powerful software to create the most widely used GIS. Their most recent development is the TIGRIS Object Oriented GIS Environment. This product has been in development since at least 1988 and has yet to be released as a production GIS. It supports a full object oriented data model and its processing uses an object hierarchy to efficiently traverse the database.

3.3 Terrain databases

As explained above, a terrain database for simulation is a collection of data which models some features of the earth. Typically a terrain database will contain representations of the ground and objects that are on the ground like houses, buildings, trees, lakes, rocks, roads and telephone poles. Also included would be subtle aspects of the earth. For example there might be a salt water lake which is located next to a desert which is located next to a mountain with a snowcap. Not only do these databases contain features of the earth but they may contain models of dynamic objects such as vehicles and aircraft.

Although terrain databases are used in simulators and their function is to provide a limited form of spatial data management, they are very different in scope and function from GIS. Terrain databases process simple queries at very high speed and are not accessible as a general data repository. Instead they have restricted access and are static. GIS systems are able to process generic non-trivial queries, and they are not required to perform at high speeds.

To construct a terrain database there is usually some database modeling tool. Here a user will take terrain information from multiple sources and, using software, automatically combine features into one data module. The user will then manipulate and add more features until the database meets specifications. Finally there will be software which takes the constructed data module and transforms it into binary data which an image generator can read and process.

Many of today's simulators require that multiple players participate in the same scenario. Networking of simulators is not a new problem, but as requirements for higher fidelity image generators increase, network traffic also increases. Some of the problems can be solved with faster hardware, but it is important to look at the semantic relationship between players as an area where substantial reduction in network traffic can take place [Hughes 90].

3.3.1 Sample Terrain Databases

The following are two examples of terrain databases. The database from the SIMNET trainers and the ESIG 500 image generator will be used to describe typical terrain databases which are in use today. First, the modeling process will be shown, followed by a description of each database architecture. This should give a detailed view of current terrain database technology and some implementations.

3.3.2 ESIG 500 Databases

The Evans and Sutherland ESIG 500 Image Generator [ESIG 90] [Moshel 90] is a real-time image generator which supplies high resolution and high fidelity graphics for use in simulators. It is independent of any particular simulator and can be configured with multiple dependent channels. The database design tool is the E&S Metafile Editor (ESED) and runs on a workstation which is networked to the image generator.

The database editor supplies information to the ESED program about the construction of the database. This information is stored in a hierarchical structure called a **Metafile**. A Metafile is a collection of **polygons, objects, cells** and **meshes**. The Metafile is the user configurable database which is presented to the image generator. This may not be the only representation of the data. Once compiled into binary files, the database is reconfigured for optimal performance. The presentation here is concentrated on the user configurable database. Figure 3.3 is a pictorial view of a sample Metafile.

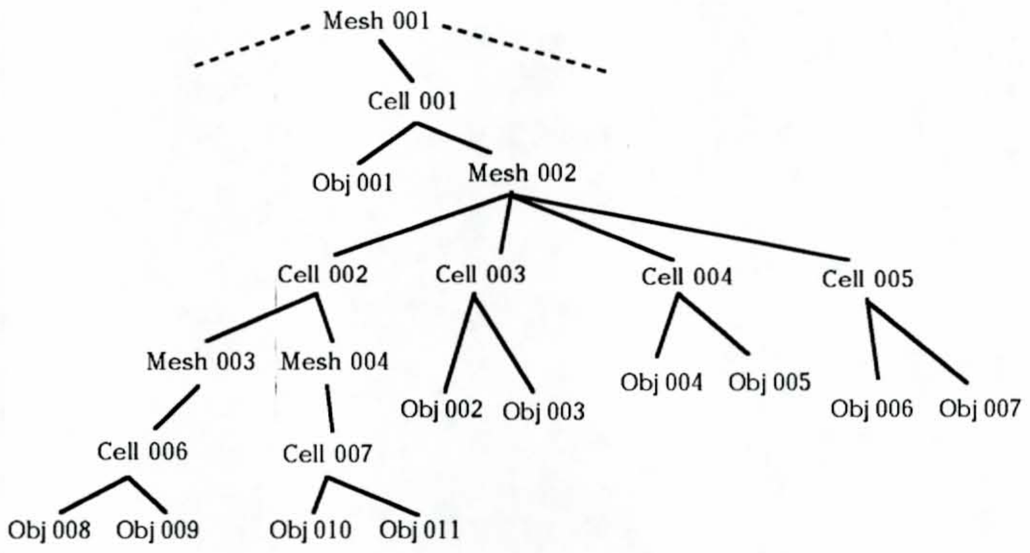

Fig. 3.4 Metafile structure

A polygon is a three or four sided, planar, convex geometric construct. A polygon has a front and back side which can be determined by its normal vector.

Included in the collection is a string of lights. Lights are similar to polygons except that they define light sources and are not planar. Objects are sets which contain polygons, and lights and are used to define entities such as planes, buildings and vehicles. A cell is a database record which contains two objects, one or two meshes or one of each. Cells are used by the system to determine levels of detail (LOD) of objects. The LOD of is a perception trick which helps reduce the amount of data necessary to describe an object. When an object is located far away from a viewer, the image generator will render a less complex description of the object. The polygon count in the most detailed object (close to the viewer) can be 100 times more than the least detailed object (farthest from the viewer). The final part of the Metafile is the mesh. The mesh is a collection of cells and are of two kinds: either **priority** or **special effect meshes**. Priority meshes define which objects will have rendering priority. Special effect meshes are used for animations and collisions.

Once the user defines the database using the ESED tool, it must be compiled with the Metafile Compiler into binary code. Next the Parcel Generator creates a flyable database which combined with the World Description file gives the Real-time system.

The most important part of the database to note is the model. It is hierarchical but not object oriented. It only contains the information necessary for creating images. It does not provide any way to interactively change the database once it is running on the image generator.

3.3.3 SIMNET Database Organization

The SIMNET database is generated using the S1000 tool. The database is constructed by means of an interactive graphics program. Using this tool the user can create terrain from DMA DTED and DFAD (Digital Terrain Elevation Data, Digital Feature Analysis Data) [DMA 86]. Once the terrain has been created, the user can define objects which will be on the terrain such as trees, tree lines, canopies, roads and rivers. These objects are placed on the terrain by using an (x, y) coordinate. Their altitude is obtained from the elevation of the terrain at that (x, y).

Once the database has been built with S1000, it is compiled to a binary file and loaded onto the image generator. The compiled database has additional information which is needed by the image generator. This data is never seen by the database modeler. Figure 3.5 is the hierarchical structure of the SIMNET terrain database as defined by the user. The main structure is kept inside of the image generator, but when compiled, more information is added.

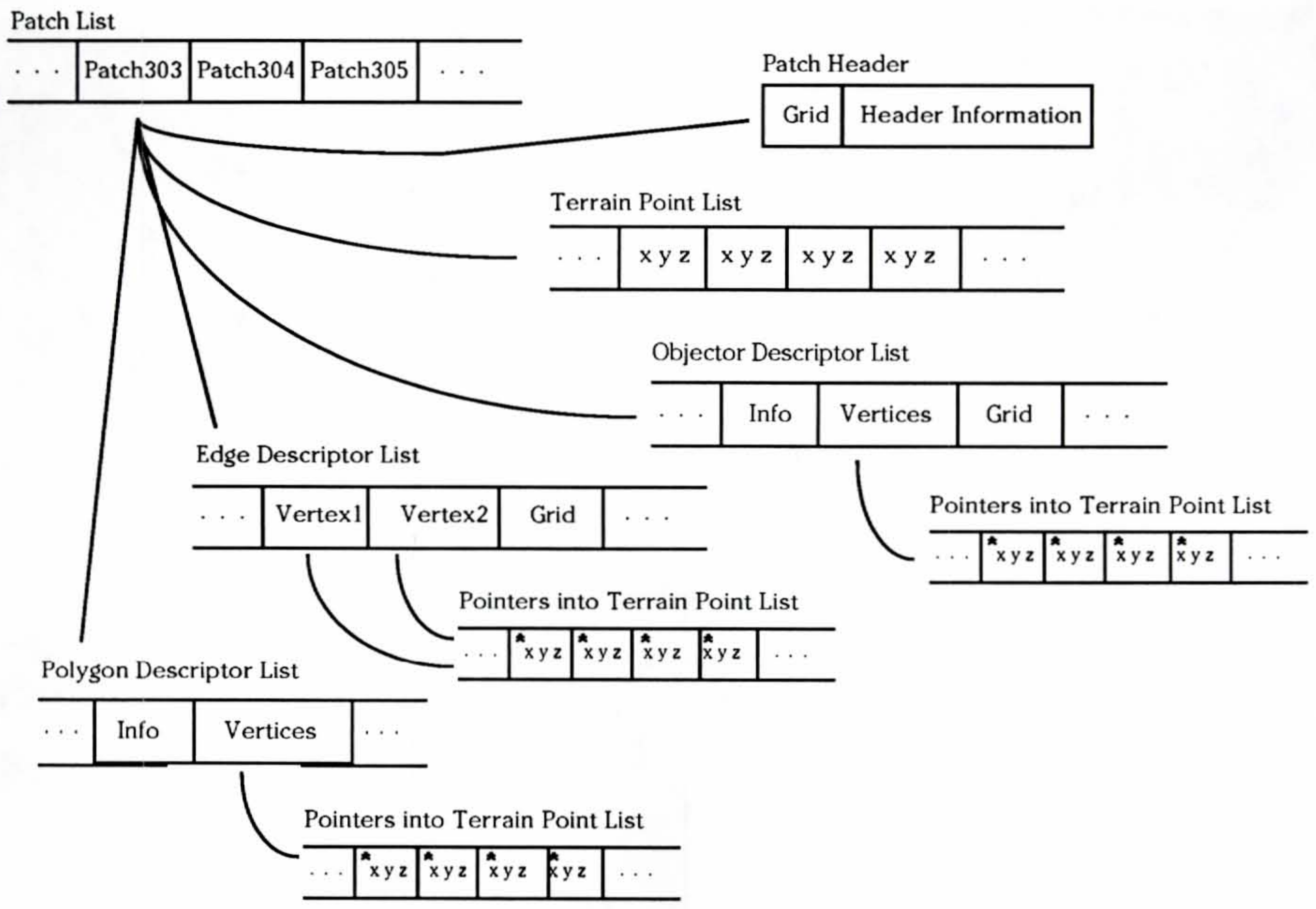

Fig. 3.5 SIMNET database organization

An interesting aspect of the SIMNET database is the organization of spatial data. A **patch** describes a 500m x 500m area which includes **quads**, **polygons** and **objects**. Each quad encompasses a 125m x 125m area and has a pointer to the polygon which covers it and to any objects which are located in its area. A single polygon may cover multiple quads. The list of quads in a patch is ordered to facilitate easy access. Each quad has a unique index which can be derived from a simple formula. This makes traversal of patch structure fairly computationally inexpensive. In addition to patches, the compiler adds another level in the hierarchy called **load modules**. This structure is used by the image generator to take advantage of the placement of patches on the database disk. This enables the database traversal to be quite efficient.

There is an attempt in the SIMNET database to efficiently model terrain without losing accuracy. There has been some additional work on using this database for more specific queries. Stanzione [Stanzione 89] has proposed additional structures which will work in conjunction with the current database model. This new format will allow the processing of detailed terrain data as well as objects. It will support rapid search techniques and storage efficiency. Their approach is to use a quad tree structure to define the terrain and the objects that are associated with it.

3.4 Summary

Chapter 2 and 3 have been a review of the current technologies. Object oriented design is a promising technology which will help better organize and encapsulate the terrain database. Object oriented database models have provided insight into the world of efficient storage and traversal techniques. Simulator terrain databases have provided a base to start with. They define the upper limit of the amount of data necessary to obtain a good visual picture. Finally, spatial data management has looked at efficient methods for processing three dimensional data in an object oriented database.

4 Object Oriented Terrain Database

This chapter is a detailed description of the OOTDB that was developed as part of this project. First there will be a description of the general format of the database followed by the specific implementation of the Terrain class and its subclasses. Throughout this chapter there will be an example which will evolve and will continue into chapter 5.

4.1 Overview of OOTDB

The OOTDB was designed to be used in dynamic simulations. The underlying structure of the database is a Smalltalk-80 based object oriented hierarchy. It is made up of classes, objects and their respective hierarchies. The main goal of the terrain database is to concisely describe a region of the earth using object oriented design and spatial data management. This format follows an object oriented design where the data model will be a class hierarchy with the most abstract classes at the top and the most general classes at the bottom of the hierarchy.

To give this project credibility, it was necessary to find some database which is in use today. Since the SIMNET tank simulators were available to use, the database of Fort Knox, Kentucky was employed as the main OOTDB file.

The coordinate system used is the Euclidean three dimensional space. Since the terrain being represented is a real place, its description is in longitude and latitude pairs (lat/long). The header of the data file shows the exact location of the bottom left patch, and thereafter each patch has its own (lat/long) location. Each polygon in one patch is numbered according to its relative location in that patch. All measurements are in meters and the spacing between elevation posts is approximately 125m. Polygons connect the elevation posts and are either three or four sided.

4.2 The Root Class Terrain

The structure of the terrain database has a special root class called **Terrain**. This class will contain all of the necessary attributes and actions to support a minimal terrain database. Each subclass of Terrain is responsible for conforming to the protocol that has been established. In addition, classes may add to and extend the protocol to better represent their intended attribute and function domain. This means that any specialization of Terrain must, either by inheritance or redefinition, respond to all of the methods that have been established at higher levels in the class hierarchy. This structure implies that there will be a single class hierarchy and possibly multiple independent part-whole hierarchies.

At the highest level, Terrain is a special class because it must establish the structure of the basic database. Each instance variable describes a particular aspect of the terrain. Not only will this basic data structure describe the natural physical structure of the terrain, but it will also have pointers to man-made structures of the terrain. For example, the class Terrain will describe the surface of the ground as well as any bridges, roads, houses, and moving vehicles which are located on or near the ground.

4.2.1 Class Region and Description of Terrain's State

At some time during a simulation, it may be necessary to instantiate pieces of terrain which will not belong to any specific concrete class. It is

```
Class Terrain
        Instance Variables
                surface: list of instances of class Polygons
                components: list of instances of any Terrain class
                overlaps: list of instances of class Regions
                partOf: list of instances of class Regions
                occupiedBy: list of instances of class SimulationObjects
                extent: an instance of class EdgeList
        Methods
                addObject: anObject
                elevationAt: aPoint
                extent
                addSurface: aTerrainObject
                entirelyEnclosedIn: anObject
                craterAt: aPoint withForce: aFloat
                elevationAt: aPoint put: anotherPoint
                intersects: anObject
                overlapsAnyone
                visibilityFrom: point1 to: point2
                root
                receiveRain: aFloat
```

not permitted to make specific instances of Terrain because it is an abstract class, so we will define another class called **Region**, which will be a concrete subclass of Terrain. Formally, the class Terrain and its associated concrete class Region will be defined in section 4.2.2.

4.2.2 Class Terrain's Instance Variables

Each of the instance variables and methods listed above have distinct meanings and functions which support the operation of the terrain database. Their implementation is not known outside this class, but it is useful to look at the details of each attribute to better understand the internal processing of data. The following detailed list of each attribute and function describes the responsibilities and scope of the base class Terrain.

surface

List of instances of the class Polygon. All polygons collectively describe the physical formation of the ground. There will only be one object associated with each actual instance of a polygon, but each instance of Region will

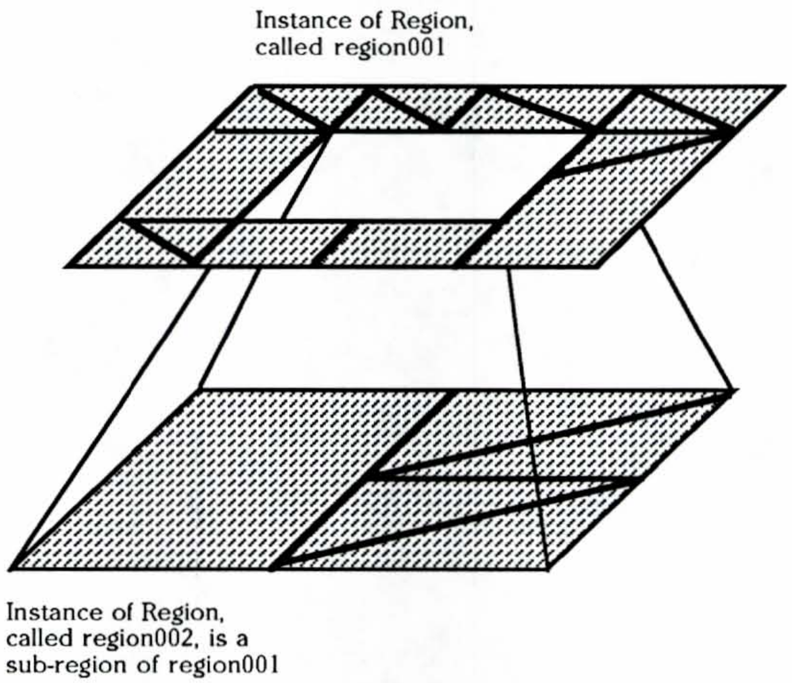

Fig. 4.1 Example of region object which has some polygons in a sub-region

contain pointers to the polygons in this list. Each instance of Region will contain only those polygons which it can directly describe. A Region may have sub-Regions which themselves have polygons. Only those polygons which are in the parent region and not in the sub-Region will remain in the surface list. Figure 4.1 shows an example of this separation of polygons between Regions and their sub-Regions.

components
List of instances of any Terrain class. Each object in this list has a physical location within the terrain database. Each component is entirely enclosed in the object which owns this list. If an object which is an instance of class Tree is in the components list of some Region, then the location of the Tree is completely within the area of that Region.

overlaps
List of instances of the class Region. Each object in this list has a physical location within the terrain database. Each object in the overlaps list is partially enclosed in the object which owns this list. This part of the database is important to reduce the complexity of the searches which take place as part of the query processing. Because all members of this list overlap in area with other objects, it can be thought of as the list of neighbors for any one instance of Region. When an object is added to the database, its overlap list is computed and stored. This provides a convenient way for the query processing to find local neighbors as well as large regions.

partOf
List of instances of the class Region. This list can be thought of as pointers upward in the part-whole hierarchy. If a Tree is in the components list of a Region, then the Region will be in the Tree's partOf list. This part of the database is used to find the root of the part-whole hierarchy as well as finding larger areas of the database. Each Region describes some area of the terrain. Its parent Region will describe some larger encompassing area. This is also useful for finding the owner Region of some terrain feature like a tree or house. Additionally, this list may contain more than one instance of Region, meaning that the particular object belongs to more than one leaf of the part-whole hierarchy.

occupiedBy
List of instances of the class SimulationObject. These objects are contained either partially or wholly within the area of the surface instance variable. This list is used to maintain the location of any transient objects. These objects are not necessarily part of the database class hierarchy, but it is the responsibility of the data manager to keep track of their locations. For example, a tank may be driving over the terrain. Its location can be maintained from its position in the part-whole hierarchy.

extent
Instance of the class EdgeList. The EdgeList class is implemented as a doubly connected edge list (DCEL) [Perparata 85] if the region is an area or a line. If the object is a point, its implementation is still the DCEL, but any access to that spatial structure will result in point-only operations. The main use of this data structure is in display and area comparison methods. See section 4.5 for more details of class EdgeList and the DCEL.

4.2.3 Class Terrain's Public Methods

All methods which belong to subclasses of Terrain must conform to the standard use of the instance variables defined above. Abstract subclasses of Terrain will probably add more instance variables as well as methods, and the same rules will apply for any additional subclasses. Not only must each specialization conform to the use of instance variables, but each must also conform to the use of methods. The methods listed below are an example set which one would use in the management of a terrain data base. The following definitions are the protocol which all objects must use when communicating with the terrain database. Additional methods are local to class Terrain and will not be described here.

addObject: anObject
Adds anObject to the components list. This message may be sent to the root of the database, or a specific component to be added to the part-whole hierarchy. This is one of the main working methods in the class Terrain. It sets as many of the instance variables of the new object as possible. First the partOf method is set using the receiver. Next, the components and

overlaps instance variables are determined using a combination of the local methods **entirelyEnclosedIn** and **overlapsAnyone**. At this point, the extent attribute is used to determine the spatial relationship between objects. The surface variable is set prior to execution of this method.

elevationAt: aPoint
Answers the elevation above sea level at aPoint on the terrain. This message is sent to an object which may the the root of the part-whole hierarchy. First, the hierarchy is searched to find which objects contains aPoint. To get the exact elevation, the plane equation of the polygon which contains aPoint is solved for its z value. See section 4.4.1 for more discussion.

extent
Answers the DCEL describing the enclosing area.

addSurface: aTerrainObject
Sets the surface instance variable of the receiver. This method is used when terrain objects are first built. First an object is constructed from disk file data. It is instantiated as a Region and polygons are added. Finally the object must be added to the database part-whole hierarchy, and this method is used to build the surface instance variable. aTerrainObject is the region object which was built from the disk file.

entirelyEnclosedIn: anObject
Answers true if anObject is entirely enclosed in the receiver. This method determines if an object is entirely enclosed within the receiver.

craterAt: aPoint **withForce**: aFloat
Tells the database to create a crater at aPoint with the explosive force of aFloat. The crater is centered at aPoint with a radius which is proportional to the number given, with 1 meter of radius equal to 1 unit of force. NOTE: Although this method is not yet implemented, all of the functionality is available in the database to complete this transaction. The methods which will determine the actual structure of the crater are not the responsibility of the database, but of the physical modeling processor section of a simulator.

The database's responsibility is to maintain the data, not be the dynamics and physics sections. Terrain modification is discussed in section 4.4.3.

elevationAt: aPoint **put**: anotherPoint
Tell the database to change the elevation of aPoint. This method has the restriction that aPoint must be an existing elevation post. It is similar in implementation to **elevationAt**, except that once the object which contains aPoint is located, one of its vertices is modified accordingly. This method would be used by an application specific method such as **craterAt**: aPoint **withForce**: aFloat.

intersects: anObject
Return true if anObject's physical area intersects the receiver. This method, using the hierarchy, determines if anObject overlaps the receiver by at least one point in space. The method to determine overlapping structures is contained in the EdgeList class. The methods there are based on the DCEL and some simple intersection algorithms which come from basic geometry. See section 4.5 for more details about the EdgeList class.

visibilityFrom: point1 **to**: point2
Return true if there is a visible path from point1 to point2. This is one of the most interesting methods in this project. Further discussion of intervisibility can be found in section 4.4.2. To determine if one point can see another, a ray casting algorithm was used [Glassner 89, 90]. A ray is cast from point1 to point2. If any object obstructs that ray, then there is no visibility. For this method to properly work, all objects must have some polygonal representation. In this case, because of the format of the SIMNET database, objects such as trees and bushes will not participate in the intervisibility query.

root
Return the root of the part-whole hierarchy. Using the partOf instance variable, this method does an upward traversal of the part-whole hierarchy to find the object which is located at the top.

receiveRain: aFloat
Modify the state variable of the object which receives this message. The parameter aFloat represents the amount in cubic meters of water that have fallen. As implemented, this method requires that subclasses redefine the action of receiving rain.

overlapsAnyone
Return true if the receiver overlaps any object in the part-whole hierarchy. This method uses method root to determine which object is at the top of the part-whole hierarchy, and then sends the root object the intersects message.

4.3 Using The OOTDB Design To Create A Sample Database

Both the instance variables and methods make up the formal protocol of the Terrain class. To create a terrain database, the minimum class hierarchy must include classes Terrain and Region. Since Terrain is an abstract class and Region is concrete, there can only be instances of Region. To create depth in the database, there must be a natural breakdown of the world being modeled.

To see the usefulness of this methodology, think of the Terrain and Region classes as a library. When the programmer wants to create a new database format, he must use the class library just as he would use a C library. The programmer would create new classes which specialize the library to meet specific needs. He will rely on the classes to perform specific tasks without questions about implementation, and the classes will require any new classes to follow the predefined base protocol.

Figure 4.2 illustrates only one possible combination of abstract and concrete classes. This class hierarchy is loosely based on the SIMNET database format. Because SIMNET data was the only production image generator data used, the example terrain hierarchy resembles the actual data in names and in structures.

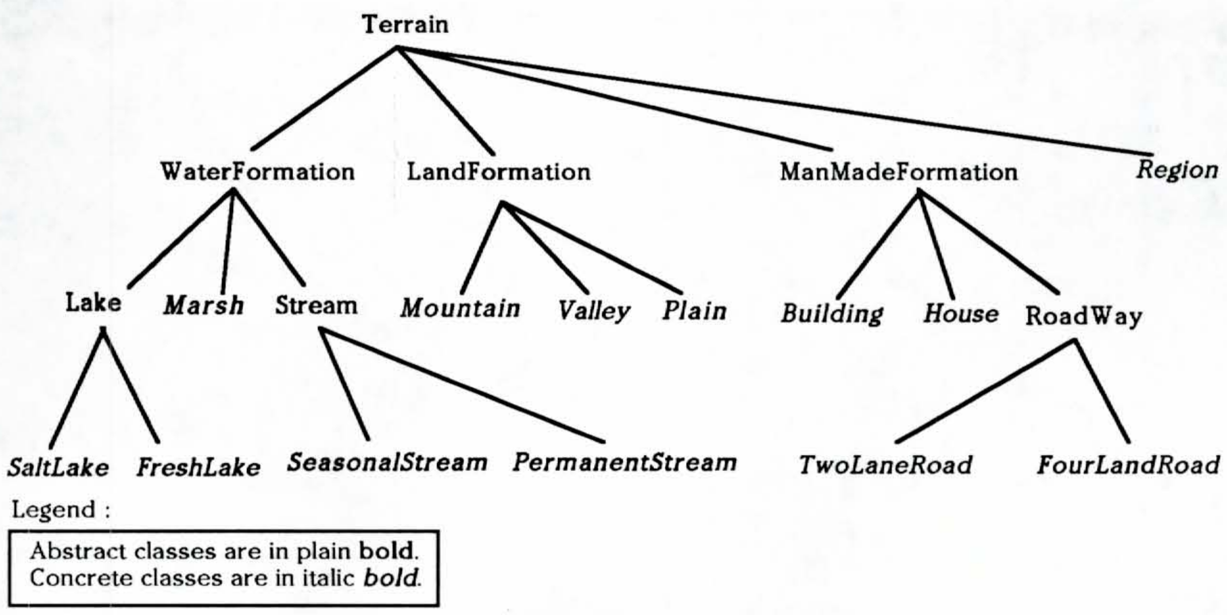

Fig. 4.2 An example class hierarchy based on the SIMNET data model

The following class definitions will demonstrate how the class hierarchy specializes the data definition. Each class has a specific purpose and was designed based on the specification of the original data.

```
Class WaterFormation
    Instance Variables
        totalVolume: an instance of class Float
    Methods
        volumeAt: aPoint

Class Stream
    Methods
        directionOfFlow: aDirection

Class SeasonalStream
    Instance Variables
        activeMonths: list of instances of class Integer
    Methods
        atPeak
        nowActive

Class LandFormation
    Instance Variables
        soilType: an instance of class Integer
    Methods
        tellTrafficabilityAt: aPoint
```

Class **Mountain**
Instance Variables
percentSnowCoverage: an instance of class Float
Methods
isPassable: aPath

Class **ManMadeFormation**
Instance Variables
materials: list of instances of class MaterialType

Class **Roadway**
Instance Variables
name: an instance of class String
speedLimit: an instance of class Integer
easementWidth: an instance of class Float
Methods
vehiclesPerHourAt: anInteger

Class **FourLaneRoad**
Instance Variables
medianType: an instance of class Integer

The instance variable trafficability is an attribute of soil. It describes how well vehicles can pass over a certain piece of earth. The following

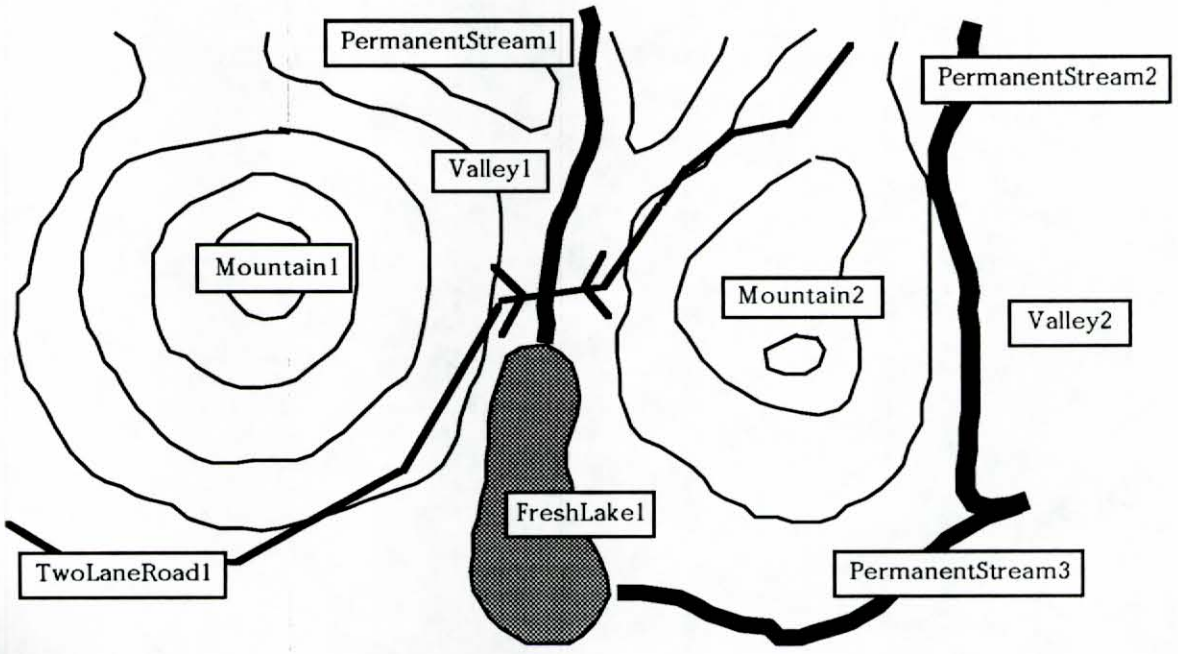

Fig. 4.3 A map with labeled areas which represent database objects

diagram is a map of an example terrain and the specific objects that will be in the database.

For the data that is represented in Fig. 4.3 to become a database, an instantiation of Region must be created. One object must be created which represents the entire land region. Other objects are created to represent each of the distinct land features such as the lake and roads, as well as the non-distinct mountains and plains. Once these objects are created, they are placed into the part-whole hierarchy.

Figure 4.4 is the database's part-whole hierarchy, with the root object being named Region1. This figure shows the relationship between objects contained in the components and overlaps instance variables in the class Terrain. Even though each member of components is completely enclosed in its parent's bounding area, it still may be possible for a region to be in more than one component list. The object FreshLake1 is a member of the components list of both Mountain1 and Valley1.

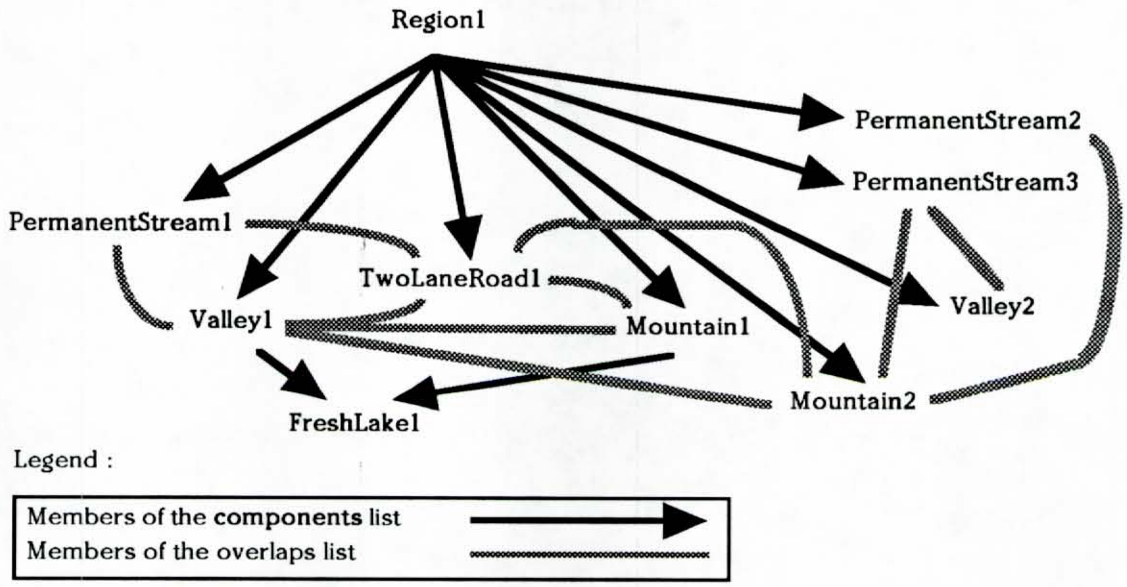

Fig. 4.4 A part-whole hierarchy showing the relationship between instance variables of the class Terrain

4.4 Query Processing

Now that the terrain database has been defined, we must look at how this database will function and how it will respond to those queries that will be necessary to support simulator activities. Efficiency, intervisibility and terrain modification are important areas of consideration when discussing query processing.

In the discussion above it was noted that sending messages to objects in an object oriented language will cause some action which is internal to that object. Also discussed was query processing in an object oriented database. These are two different operations that are essentially the same in nature. An object oriented database will process queries by sending database objects messages. Once the user has queried the database, the query processor will determine which object is to receive the message, then execute the appropriate method in that object. An object oriented language will process a message in a similar manner, decoding the message and executing code of an object. Because the OOTDB has an implementation in Smalltalk-80, queries are the same as sending message to objects. Because the OOTDB was designed independent of any implementation, generic query processing will be left as a black hole to be filled in by some application.

Each class has the responsibility of responding to queries in an appropriate fashion according to its predefined course of action. For example, Valley1 will respond to **receiveRain** by substantially changing its trafficability, as compared to FreshLake1, which would just add to its volume and perhaps extent. Depending on the recent history of FreshLake1, enough rainfall would completely change its trafficability and appearance. Notice that there are two distinct phenomena that occur simultaneously; first, there is the uniform requirement imposed by inheritance that all subclasses of Terrain respond to **receiveRain**. Second, there are the non-uniform reactions by the various subclasses of Terrain.

4.4.1 Efficiency

The efficiency of query processing is very important in a real-time simulator. Even if the response time is slow due to a particular implementation (e.g. Smalltalk-80), the query processing must be algorithmically efficient so that a re-hosting will yield a usable system. The information of the database is contained in the part-whole hierarchy, and the methods of the classes are the executable parts of the queries. This implies that a careful working relationship of these system components is necessary.

In previous discussion of spatial data and query processing, Antony [Antony 88] mentioned that there might be a useful combination of spatial data processing based on quad-trees and object oriented part-whole hierarchies. The promising merger of these two technologies was tempting enough to at least try to justify the OOTDB as meeting this criteria. Most of the part-whole hierarchy tree traversal is accomplished by the use of an in-order search of the component's list located in the class Terrain. Because this list contains only those objects which are completely enclosed in their parent objects, each search will find the correct object without losing its place in the hierarchy. So in fact there is some relationship being shown here between spatial data and object oriented design. The difference is that the OOTDB has its spatial data embedded into the part-whole hierarchy whereas Antony suggests it should exist in parallel.

The following code section is taken from the implementation of the OOTDB which is written in Smalltalk-80. There is one local variable, elevation, located between the vertical bars. The method **elevationAt**: aPoint is a traversal of the part-whole hierarchy. Other methods such as **elevationAt**: aPoint **put**: anotherPoint and **intersects**: anObject are very similar in their implementation.

```
elevationAt: aPoint
      | elevation |
      (extent containsPoint: aPoint)
            ifTrue: [
                  surface do: [:each|
                        (each containsPoint: aPoint) ifTrue: [
                              ^each elevationAt: aPoint]
                  ]
            ]
            components do: [:each|
                  elevation <- each elevationAt: aPoint.
                  elevation isNill ifFalse: [^elevation]
            ]
```

This method first searches its own local space to see if it contains aPoint. If so, the correct surface polygon which contains aPoint is found. Then the polygon is asked to exactly determine the elevation. If aPoint cannot be found in any of the surface polygons, then all of the components of the receiver are searched. This process is continued until the polygon which contains aPoint is found.

When this message is sent to the root instance of Region, called Region1, all of the surface polygons must be searched. Remember, if the components list is not empty, then there may not be any polygons in Region1's surface list. This is why we must search all of the members of the components list.

The most important aspect of this query is the structure of the part-whole hierarchy. Because the components list can be thought of as a tree, we would like to have as balanced a tree as possible, with the constraint that this list represent the natural structure of the terrain. The above query is correct in either case, but is most efficient when the tree is exactly balanced.

4.4.2 Intervisibility

The question of intervisibility is important to the world of simulators, and in particular, automated simulation objects such as so-called "semi-automated opposing forces" (SAF). These automatic players must be able to query the environment, to examine their surroundings, and to determine what they can "see". The question "Can object Q be seen from a height of z at location (x,y)?" asked of the terrain database is particularly interesting and difficult.

The most basic of answers involves geometric calculations concerning line of sight. Is there any one object blocking the view path from one point to another? This problem has been solved many times over in ray tracing [Glassner 89, 90]. This concept then can be extended for line of sight from one single point to another and from one single point to an object which has an area. This would involve taking into account all the bounding edges of the object. This again can be solved geometrically and for certain configurations of terrain, can be solved efficiently. [Cole 89] showed that for certain configurations of polyhedral terrains and viewing points, the complexity of computing the visibility of one point to another can be as good as $O(\log^2 n)$ time for ray shooting type queries (n is the number of faces).

The most difficult questions to answer are when the line of sight is through some semi-transparent object. For example, if there is a line of trees between two objects, can they see each other? There are a few possible paths to solutions for this question. First, if the answer depends on the image from an image generator, then the only assistance needed from the database is scene geometry. Second, there is always the perceptual question, "If object 1 can physically see a very small part of object 2, does object 1 recognize object 2 as object 2 and not something else?" The answer to this question should come from the internal processing of object1 itself. This implies that the database will be limited to geometrical intervisibility queries based on the terrain only.

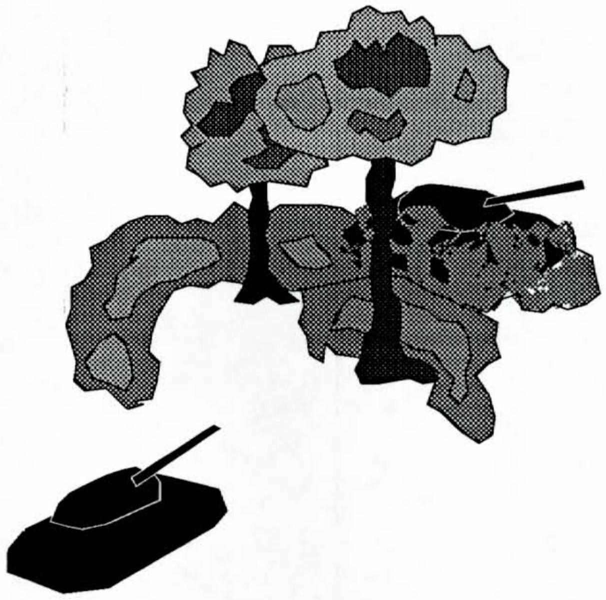

Fig. 4.5 Can both tanks see each other?

Because this is a very interesting question posed to a terrain database, it was included as part of the standard protocol for the OOTDB. This implementation does indeed find if a line of sight exists between two points based on the layout of the terrain polygons. The algorithm above comes straight from simple ray casting [Glasner 89]. It was important to show that these types of complicated queries can be included and supported in this type of database. The algorithm was written so that any subclass of Terrain may use it without regard to its implementation. It also can be modified to support an individual class's line of sight specific processing. Specifically, the line "loop on all polygons" is obviously not the most efficient

```
visibilityFrom: aPoint to: anotherPoint
    Calculate the ray direction from aPoint to anotherPoint
    Loop on all polygons
        Ignore polygons which are parallel to the ray
        Find distance from ray origin to polygonal plane
        If distance is closest so far then
            Calculate point of intersection of ray and polygon
            Project intersection point using Jordans curve
            Remember this point if it is closest
    If the closest distance is less than the distance from aPoint to
    anotherPoint, then there is no visibility
```

way to traverse the database. It will be more efficient to use the part-whole hierarchy to determine the spatial relationships between objects to better traverse the database.

4.4.3 Terrain Modification

This is one of the most important commands in the terrain database. This function will be included to support such simulator activities as terrain deformation, erosion, and any other actions which will affect the elevation of the ground. It is not the responsibility of the database to compute the physics of how the earth will be disturbed, but it does have the responsibility of remembering how it was changed. Also it must inform any simulation players about the changes to the terrain, especially those players who must supply images.

There has been work done on automatic terrain deformation. There are two areas where research has been concentrated. First, automatic level of detailing may be a possible area where algorithms may be found [Scarlatos 90]. Also, work done by [Provost 90] may lead to high speed generation of microterrain. There, large polygons are automatically broken into smaller ones when terrain deformation is needed. Then vehicles such as bulldozers are free to disturb the ground as they please.

For the OOTDB to support such activities, it is necessary to include some specific algorithms to support terrain changes. The proper assignment of responsibility is crucial to the efficient operation of the data manager. This division of labor occurs at the point between physics and data storage. Imagine some infinitely powerful soil dynamics processor that could take the definition of the terrain and determine its new shape if any deformation occurs. This machine would be independent of any storage techniques, except for its own local memory. It would then register the terrain changes with the data manager.

The OOTDB supports addition and deletion of polygons. The algorithm used to locate target polygons is almost exactly the same as the one described above in the elevationAt: aPoint algorithm. Simply changing polygons does not require any message propagation to inform other regions of the polygonal changes. But message are sent to all objects which reside on the changed polygons. Access to local objects can be found in the components, overlaps and simulationObject instance variables. This is because all polygons are stored only once in the database, and pointers to the polygons from objects are the only access to them. If new polygons are added to the database, insertion takes place in the same manner as reading from a disk file.

4.5 Additional Classes Which Support the OOTDB

There are four additional classes which are an intimate part of the OOTDB although they are not part of the Terrain class hierarchy. These classes represent data structures that would be found in any spatial data system. They were all put into classes because Smalltalk-80 does not support them. The classes are EdgeList, Edge, Points, Polygon.

4.5.1 Class EdgeList

The class EdgeList encapsulates the doubly connected edge list (DCEL) data structure. The class definition is :

Class **EdgeList**
- Instance Variables
  - **listOfEdges**: list of instances of class Edge
  - **origin**: an instance of class Point
  - **corner**: an instance of class Point
- Methods
  - **intersects**: anEdgeList

This class represents the main spatial representation which exists in the database. Since the SIMNET data is three dimensional, the DCEL is an excellent method of representing the data. It is mainly used to describe the enclosing area of a region. A region is a collection of contiguous polygons. A

DCEL can be used to represent all of the edges of polygons surrounding the region. This edge list encloses either a convex or concave area, and there exist algorithms which can determine if any two DCEL structures intersect.

To test to see if two regions intersect, it is only necessary to test the enclosure of the edge lists without their elevations. This is because polygons that are common to more than one region will have the same edge representation. The algorithm used is a simple loop which checks to see if any of the edge points of one region are contained in the other region. If even one point passes this test, then the regions intersect. This is an $O(n^2)$ algorithm, it is one of the best known [Preparata 85]

4.5.2 Class Edge

This class is a simple structure representation of an edge. An edge is defined as the line connecting two three dimensional points. Routines are supplied which test the equality and intersection of two edges. This class is mainly used in the EdgeList class. The definition of class Edge is :

```
Class Edge
        Instance Variables
                point1: an instance of class Points
                point2: an instance of class Points
        Methods
                intersects: anEdge
```

4.5.3 Class Points

Smalltalk-80 does not contain any representations of three dimensional points. The class Points was added to Smalltalk-80's graphics library as a general tool. It has the same protocol as the Smalltalk-80 class Point, which represents two dimensional Cartesian points.

4.5.4 Class Polygon

The class Polygon is the main spatial data structure of the OOTDB. It is used in many places in the database to support graphical display. The structure of Polygon is simple: it contains a list of points which are the vertices of the polygon. The class definition is :

Class **Polygon**
    Instance Variables
        listOfVerticies: list (3 or 4) of instances of class Points
    Methods
        edgeList
        interiorPoint
        containsPoint: aPoint
        determineZAt: aPoint
        display

This data structure was encapsulated into the class hierarchy class because it was used many times during the design of the database. Although these operations are simple to implement, it was easier to remove the attributes and actions and place them in a separate class.

5 Object Oriented Terrain Database Editor

During the design of the OOTDB, it became necessary to develop tools that would help test and verify its proper operation. In addition, there are certain functions that are needed to support the main database activities but are not reasonably assignable to a class of the terrain hierarchy. The **Object Oriented Terrain Database Editor** (OOTDBE) was created to help in the building and learning process. With this editor, a user can create a new database from a disk file and edit terrain features including modifying the part-whole hierarchy. There is also support for persistent storage of the database using the GemStone Object Oriented Database system.

5.1 Overview of Editor

In looking at the simulator database systems described in chapter 3, it became obvious that any good terrain database works in conjunction with other software systems. All of the systems described above are supplied with editors and compilers which give the user easy access to the raw data. The terrain editors are typically interactive graphical sessions where the user can manipulate terrain data as well as terrain features. These operations will include reformatting the polygonal structure and adding features to the terrain to better serve the needs of the image generator.

An editor was designed for this terrain database to get some of the functionality that exists in actual systems. It will let the user directly manipulate the raw SIMNET data to create the database's part-whole hierarchy. This editor is interactive and uses a graphical display which shows the user's work already completed. The basic steps in creating a database are :

- Translate existing SIMNET file (raw data)
- List all objects in the database that are not terrain polygons.
- List the part-whole hierarchy
- Add to and modify the part-whole hierarchy
- Produce a graphical display
- Send the final copy of the database to permanent storage
- Edit the database again if necessary

The main intent of the editor is to read some raw data, edit the data to make a part-whole hierarchy, see the resulting database graphically, and send the final database, which includes the classes and the part-whole hierarchy, to permanent storage. Additionally, the user may reload the database from permanent storage and continue to edit.

This application is built using the object oriented language Smalltalk-80 and the imbedded database language of GemStone called OPAL. Because of their similarity, OPAL and Smalltalk-80 can be considered the same. Smalltalk-80 was chosen because of its pure object oriented environment, its ability to rapidly build software components, its connection to the Gemstone Object Oriented Database and its user interface development environment. This editor and terrain database are able to run on any computer which can support Smalltalk-80 v2.5 and at least provide an InterNet connection to a machine which is running the GemStone database server. [Goldberg 89] [GemStone 90].

5.2 Using the OOTDBE

Using the OOTDBE is fairly simple if the user is familiar with the Smalltalk-80 interaction style. The user should be familiar with the object oriented programming environment but does not need to have any Smalltalk-80 coding experience. To work with the OOTDBE, the user must load it into Smalltalk-80. The GemStone database server must also be accessible through the Smalltalk-80 environment. About five megabytes of local disk space are also required. Refer to the manuals for Smalltalk-80 v2.5 and GemStone v1.5 for instructions on how to start both systems working together.

5.2.1 OOTDBE Classes

The OOTDBE is constructed from classes which interface the terrain database to the Smalltalk-80 windowing environment. These classes were coded in Smalltalk-80 and follow the Model-View-Controller (MVC) paradigm which is the standard programming interface to the windowing structures. The Model class contains all of the application code, the Controller class interfaces with all of the input devices and the View class handles all of the graphical displays. Most of the OOTDBE classes are subclasses of either Model, View or StandardSystemController. The ObjectList and

HierarchyList classes are instances of the pluggable view class List. For a review of the MVC and List classes, see [Goldberg 89] [Bryden 90]. The following is a list of the classes, who they inherit from, and a brief description of their function:

Class/Type/Description

DBC/Model/Read data files, maintain Object List and HierarchyList, interface to GemStone
DBCView/View/Start the OOTDBE window
DBCController/Controller/Interface to mouse and Main Menu control
DBCGraphicsView/View/Display of graphical data, contouring, display of objects, region selecting control
DBCGraphicsController/Controller/Interface to mouse and Graphics Area menu control
DBCList/Model/Pluggable List
DBCObjectList/Model/Directly manipulate ObjectList, passing data to DBC
DBCHierarchyList/Model/Directly manipulate HierarchyList data passing data to DBC

5.2.2 Starting the OOTDBE

Once the OOTDBE software is loaded into Smalltalk-80, the class DBCView must be sent the message open. The following text can be entered in a workspace window and executed :

```
DBCView open
```

This message will ask the window manager to start the methods necessary to run the editor. Once the user opens the editor window they are ready to create object oriented terrain databases.

5.3 Window Layout

The editor window is divided into four sections, each of which can be activated by moving the mouse into a specific section. In each section, a menu can be activated by depressing the yellow mouse button. Each menu will appear near the location of the cursor, and by moving the cursor, the user can highlight one of the menu choices. To select a menu choice, highlight one menu choice and then release the mouse button.

The four sections of the editor window are Object List, Hierarchy List, Graphic Area and Background Area. The following is a diagram of the editor's window.

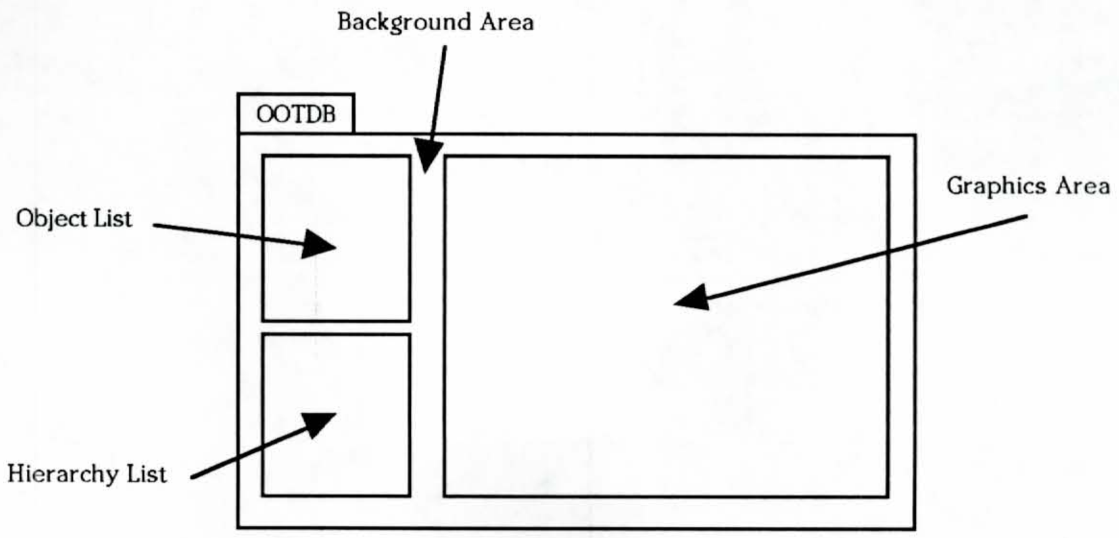

Fig. 5.1 Window layout of Object Oriented Terrain Database Editor

5.3.1 Background Area

The active menu in this area is the systems Main Menu. Using the options located here, the user can start to build a database. First, the user must read some database into memory using either of the top two menu choices. Until then, all options of all other areas are inactive. The Main Menu has the following options :

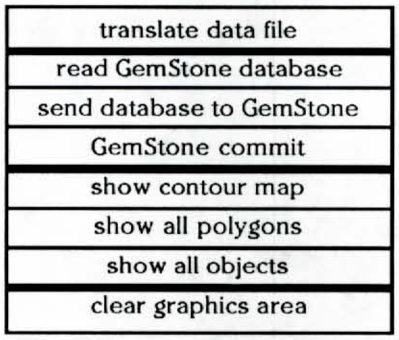

Fig. 5.2 Main Menu

translate data file
This menu option allows the user to start editing a terrain database. When this menu option is chosen, a data entry screen will appear asking the user to enter the name of the raw SIMNET data file to process. Once this file has been read, an instance of Region is created which is called Region1. This will be the root of the part-whole hierarchy. Next all of the terrain polygons in the data file are collected and placed in the surface list of Region1. This root region is added as the first entry in the Hierarchy List portion of the window. Finally, all of the objects in the raw data file are collected. For each raw object, a Smalltalk-80 object of the appropriate class is instantiated and added to the Object List. Finally, the Object List and Hierarchy List are displayed in their appropriate locations on the screen. The part-whole hierarchy is now ready to edit.

read GemStone database
This menu option is used when a database needs to be read from permanent storage. Only databases which have previously been stored can be loaded into memory using this menu choice. Once the user has chosen this option, a data entry window will appear. It will prompt the user to enter the name of a database which is listed in the top part of the entry window. Once the user enters a name, the database is loaded from GemStone into Smalltalk-80 memory. At this point, the part-whole hierarchy is now ready for editing.

send database to GemStone
This menu option is used when the user has finished creating a terrain database and wishes to make it part of the collection of permanent databases. When the user chooses this option, a data entry screen will appear asking the user the name under which the database should be stored. After entering the name, the data is copied from local Smalltalk-80 memory to GemStone.

GemStone commit
This menu option is used to tell GemStone that any operations done previously should be made permanent. The user should execute this menu option directly after saving a database in GemStone. This option is provided

to give the user a last chance to make any changes to the database before it becomes permanent.

show contour map
This menu option shows a detailed contour map of the current terrain database. When the user chooses this option, a contour map of the data will appear in the Graphics Area. This image will be combined with any image that is already existing in the Graphics Area so all images will continue to show. This is mainly used for finding hills and valleys of the terrain area. It will help the user to better distinguish different parts of the terrain [Johnston 86].

show all polygons
This menu option shows all terrain database polygons in the Graphics Area. This image will be combined with any image that is already existing in the Graphics Area so all images will continue to show. The polygons will be displayed as a plan view.

show all objects
This menu option will display all of the objects in the Object List in the Graphics Area. This image will be combined with any image that is already existing in the Graphics Area so all images will continue to show. Each object has an internal polygonal or line based representation of itself which will be displayed.

clear graphics area
This menu option will clear the Graphics Area.

Figure 5.3 is an example of an editor session. At this point in the editing session, the user has read in a SIMNET data file. No objects have been placed in the part-whole hierarchy, so the only object located in the hierarchy list is Region1. Also, the user has chosen to see the contour map, all polygons and all objects in the graphics area. Although this display is confusing to look at, it represents almost all of the visual and textual information that can be placed on the screen at one time.

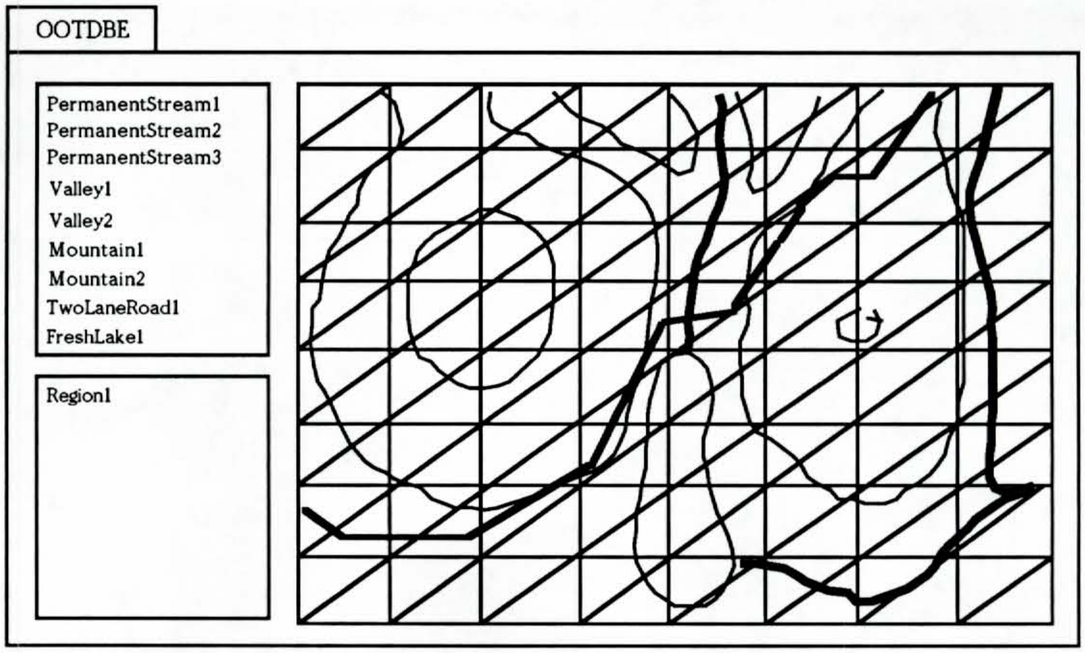

Fig. 5.3 A working session with the editor

5.3.2 Object List

The Object List contains all of the objects that are not currently in the part-whole hierarchy. It is a holding place where objects live until the user chooses one to be part of the database or to be removed completely from consideration. This list remains empty until a raw data file is processed. The only objects which are supported as part of this list are the ones defined by the Terrain class and its subclasses. Because this application has been built specifically to process SIMNET data, objects which are not part of that system are not supported here. When an object is in this list, it exists as a Smalltalk-80 object and can be manipulated as such. The menu of the Object List has the following options :

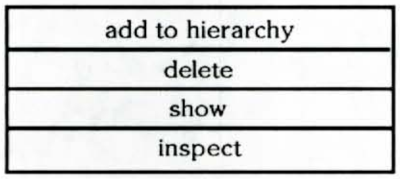

Fig. 5.4 Object List Menu

add to hierarchy

This menu option will let the user add objects from the Object List to the part-whole hierarchy displayed in the Hierarchy List. To be able to add an object to the part-whole hierarchy, the user must first choose a place in the hierarchy where the object is to be placed. This requires one object being selected with of the mouse. When a new database is loaded into memory from disk file, the only object to choose is Region1. The user may now select an object from the Object List. Then the "add to hierarchy" menu option is chosen and the object is placed in the proper position in the part-whole hierarchy. The method to add an object performs all of the necessary computations to update the instance variables of the chosen object and region.

delete

This menu option will discard the selected object from the Object List. This object is no longer part of the system.

show

This menu option will display the selected object's graphical image in the Display Area. Each object type has a different representation graphically. Most classes display their objects as polygons, but some are points and lines.

inspect

This menu option will inspect the currently selected object from the Object List. A Smalltalk-80 inspector will be presented to the user which will contain information about the object which was selected. This is the standard Smalltalk-80 inspector and more information is available in the manuals.

5.3.3 Hierarchy List

The Hierarchy List will show all of the objects in the part-whole hierarchy of the database. Only the members of components will be shown in this list. This list is presented in a hierarchical manner. Each level in the hierarchy is indented from the left side of the menu. Deeper levels are farther to the right. All of the objects here are the result of a user editing session. The menu which is active from this window is :

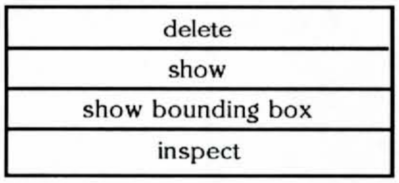

Fig. 5.5 Hierarchy List Menu

delete

This menu option will remove the object from the part-whole hierarchy. It will recursively delete all subparts which reside below the object that was selected. If any confusion is possible, the user will be asked for clarification. For example, if an object is in the components list of more than one region, and removing it from one region will remove it from other regions, the system will prompt the user to make a choice about complete removal from all regions, or only from selected regions.

show

This menu option will highlight all edges of the object selected. All objects in the database have an extent, which is a collection of edges representing the outside of the region. All edges are display and highlighted so that they stand out from other lines on the display.

show bounding box

This menu option will display the smallest box which surrounds the region that has been selected in the Hierarchy List. The box will be highlighted so that it will stand out from the other lines on the display.

inspect

This menu option functions the same as the inspect option in the Object List menu, except the object being inspected is from the Hierarchy List.

5.3.4 Graphics Area

This area is one of the most interesting aspects of the editor. The user is given the capability to edit the database graphically. The user can interactively select portions of the terrain and add them to the part-whole hierarchy. Once an area of the terrain is selected, the user can add that area as a component of some region in the database. There are certain rules that will be followed in the creation of new area, which will be explained later. There are only two menu options which are active in this area. They are :

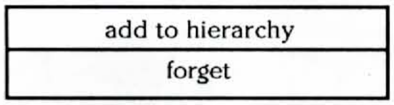

Fig. 5.6 Graphic Area Menu

permenant

add to hierarchy

This menu option adds a region of terrain to the part-whole hierarchy. The "show contour" or "show polygon" main menu option should be chosen first before adding areas to the hierarchy.

The yellow mouse button is active at any time during an editor session. First, the user will place the cursor somewhere in the Graphics Area and press the yellow button. This will mark the first point of the area to add. Thereafter, a rubberbanding line will follow the cursor until the user again presses the yellow mouse button. This action will continue until an area of the terrain has been selected. To close off the area, the blue mouse button is pressed. When the enclosing area is constructed, no lines may ever cross. The software automatically lets the user know if any illegal action has occurred.

Once the user has enclosed an area it can be added to the part-whole hierarchy. The user will then choose a place in the part-whole hierarchy at which the new region will be added. This should be similar to the actions performed when adding an object from the Object List to the Hierarchy List. Finally the user must choose the "add to hierarchy" menu option. By doing so, a new instance of Region is created and added to the Hierarchy List in the appropriate place. Again, all instance variables of the new region and of the existing region will be set.

forget

This menu option unmarks a set of features which were marked by the previous action.

The following diagram is a continuation of the previous one. It shows some depth in the Hierarchy List where the highlighted lines show the bounding area of Region2 and Region3. The objects shown in the Graphics area have been added to the Hierarchy List. It is not shown here, but TwoLaneRoad1 is in the overlaps of both Region2 and Region3. This was automatically computed when it was added from the Object List to the Hierarchy List. Also, there are other objects which overlap each other but are not shown here. All other objects are not displayed.

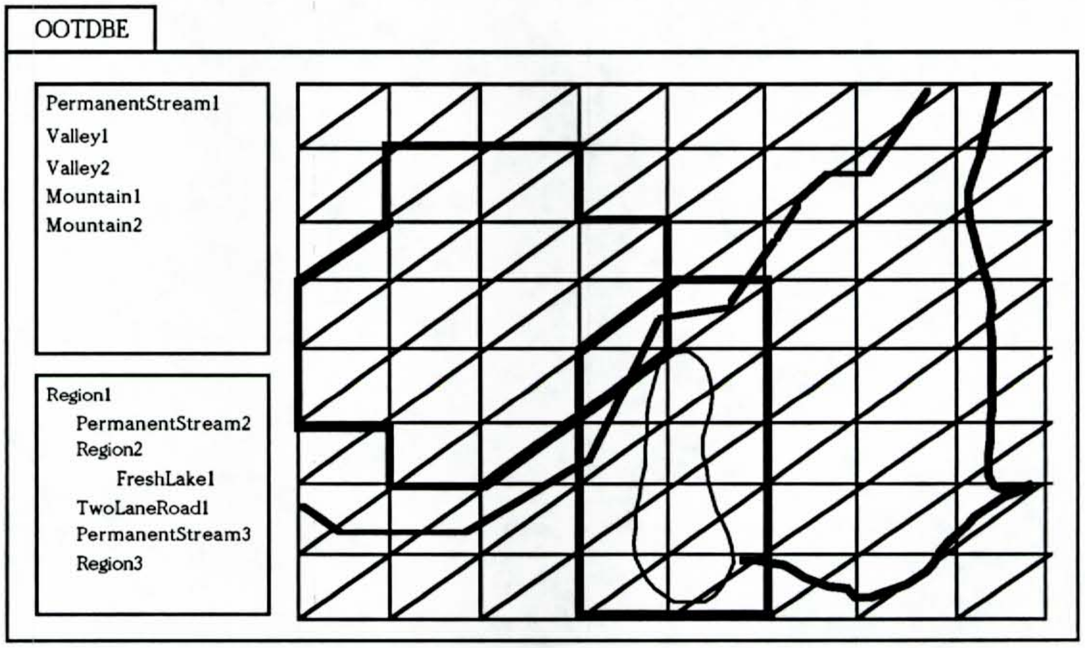

Fig. 5.7 A sample editor session

5.4 Rules For Adding an Object or Region to the Database

When an object or new region is added to the part-whole hierarchy, there are certain rules that are followed. These rules follow the guidelines set up by the Terrain class. When an object is added to the database, it must be completely enclosed in the area to which it is being added. This applies to regions as well. There are methods supplied which give the enclosing areas of objects and regions. There are also methods to do intersection tests. If this test does not pass, the object or region will not be added to the part-whole hierarchy.

After a region has been added successfully, the surface list for the new region and its enclosing region will be modified accordingly. Remember that the polygons will reside at the lowest level possible. Once the object or region has been added, the overlaps instance variable is set. A complete search of the database will reveal all parts that extend over each other. This then becomes the overlaps list.

6 An Experiment on Performance of the OOTDB

6.1 Description of the Experiment

To show that the OOTDB is a viable resource for use in the simulation community, it is necessary to conduct an experiment. This test was to determine if any conclusions could be drawn about the relative performance of the GemStone system. A database system should show improvements in speed when queries are centralized and give average results when the queries are randomized.

The following experiment involved five different terrain databases all created with the OOTDBE. There are four queries which will be sent to the database and each query will be sent a specified number of times. The timings given are the real clock.

6.2 Description of the Experimental Databases

There were five databases created and used in this experiment. Each database was created from the same raw data files but the amounts of terrain coverage by each database differs. Each database is also classified by the form of hierarchy its takes. For example, there are flat, balanced and skewed part-whole hierarchies. In addition, some of the databases contain terrain objects such as houses, trees, treelines, roadways and lakes. The following is a list of the five databases :

Database Number	Size Patches	Balanced/Flat/ Skewed	Objects or No objects
1	3x3	Balanced	Objects
2	3x3	Flat	No objects
3	2x2	Flat	No objects
4	3x3	Skewed	Objects
5	1x1	Flat	No objects

6.3 Experiment

Each database was sent four queries with both randomized and localized data points. The queries that were used were :

elevationAt: aPoint

elevationAt: aPoint **put**: anotherPoint
intersects: anObject
visibilityFrom: aPoint **to**: anotherPoint

Each query was sent to the same database which was located in the Smalltalk-80 image and in the GemStone system. When the queries were being processed in Smalltalk-80, no activity was being conducted in GemStone, and the reverse was also true.

Each query was sent 1, 5, 10, and 50 times to each database. For each query sent, the distribution of the data points was either randomized or localized. Randomized points, in the case of the elevation and visibility queries, fell anywhere inside the legal boundary of the terrain. The randomized areas which were used in the **intersects**: anObject query came from the database itself. Two objects from the terrain part-whole hierarchy were chosen at random. The localized points were statistically close together. One seed point was picked at random. This along with a random range gave an enclosure which was random in nature. All points that are localized fall within that area.

There is no data for the localized areas used in the intersects query. These areas were left out because there was no simple way to find which portions of the part-whole hierarchy were next to each other in the database. It was determined that randomized areas would be enough data for this query.

6.4 Numerical Results

The table which is located at the end of this chapter, Table 1, shows the numerical results which were obtained from conducting the experiment. All times are recorded in seconds. The queries are labeled at the top of the page while the number of repetitions are located at the side. Each database is marked with a number which corresponds to the list given in section 6.2.

6.5 Results

There are three major conclusions that were discovered about the OOTDB from the result of conducting this experiment. The first is that the terrain database functions correctly. Both the Smalltalk-80 and GemStone queries returned the same results. The second observation is that the GemStone system is slow. It does not look like GemStone will be able to be used in its present configuration in a real-time simulation. Finally this experiment shows that more experiments need to be done. There is some obvious correlation of the data from the GemStone and Smalltalk-80 queries. But because GemStone is so slow, it is almost unnecessary to make any formal evaluation of its performance.

One of the inconstancies found was in the difference in times between the random and localized points. The localized queries should have taken less time to execute than their random counterparts. The data presented in the table shows that there are many times where this was not true. The reason these times differ from the expected pattern is unknown.

6.6 Possible Reasons for Poor Performance

There are a few possible reasons why such poor results were obtained from the GemStone system. First, the configuration of the software on the workstation which houses the GemStone server may be limiting its performance in some way. This particular computer is part of a network and one of its functions is to be an administrator. It has some resources which other workstations use.

Second, this database uses three dimensional data and all of the computation uses floating point arithmetic. It is not known if the GemStone server uses the underlying floating point processor, or must process all floating point numbers in software.

Table 1. Experimental Results

		elevationAt				elevationAt:put:				intersects:				visibilityFrom:to:			
database #		Smalltalk		GemStone		Smalltalk		GemStone		Smalltalk		GemStone		Smalltalk		GemStone	
# of reps		Rand	Local	Rand	Local	Rand	Local	Rand	Local	Rand	Local	Rand	Local	Rand	Local	Rand	Local
1	1	0.35	1.0	36.5	48.41	0.20	0.48	12.42	14.96	1.11	-	46.39	-	0.78	0.84	127.63	82.49
	2	0.62	0.32	46.08	25.19	0.59	1.39	30.94	52.37	1.92	-	152.75	-	0.91	0.49	126.22	33.45
	3	0.27	0.37	26.41	29.86	0.12	0.54	7.27	24.43	0.26	-	66.36	-	0.46	1.68	49.23	52.92
	4	0.11	0.46	30.52	20.76	0.14	0.38	18.85	11.91	0.04	-	20.87	-	0.73	2.02	95.33	71.14
	5	0.13	0.08	13.24	12.95	0.05	0.07	8.82	15.43	0.10	-	53.94	-	0.14	0.20	24.27	16.43
5	1	2.20	1.74	101.14	88.65	0.64	1.77	30.08	32.48	3.10	-	434.76	-	2.45	2.30	-	-
	2	3.35	1.12	121.61	91.86	1.64	1.71	113.01	99.88	4.53	-	666.28	-	2.83	2.27	-	-
	3	2.25	0.33	69.71	32.85	0.25	1.02	37.59	82.33	1.89	-	290.93	-	1.03	0.94	-	-
	4	0.63	1.43	54.09	55.31	0.40	0.55	27.25	21.41	0.35	-	61.28	-	2.49	2.06	-	-
	5	0.33	0.29	34.26	17.44	0.71	0.35	17.99	27.62	1.71	-	80.82	-	2.88	1.06	-	-
10	1	3.80	5.57	209.28	360.35	0.36	0.93	40.34	48.67	11.67	-	1229.0	-	4.88	2.09	-	-
	2	3.32	2.75	222.64	161.54	1.75	2.41	186.85	242.76	6.27	-	1324.9	-	5.32	3.93	-	-
	3	0.73	0.75	91.04	38.33	1.96	1.23	85.12	131.80	2.33	-	571.97	-	2.02	1.59	-	-
	4	2.73	2.40	88.21	117.50	0.68	0.66	35.31	38.16	0.69	-	87.60	-	5.78	3.19	-	-
	5	1.83	0.43	36.68	30.63	0.25	0.63	20.19	56.87	1.84	-	161.75	-	1.79	1.13	-	-
50	1	11.21	12.91	1201.4	1105.8	2.41	3.02	141.00	173.96	30.53	-	4046.2	-	25.76	18.21	-	-
	2	9.28	1.74	955.27	141.55	5.34	10.25	586.56	1051.5	30.47	-	6764.7	-	24.91	16.53	-	-
	3	5.19	2.65	420.94	258.47	2.73	1.48	324.03	111.51	12.14	-	2858.4	-	10.14	7.16	-	-
	4	3.78	4.36	394.55	452.00	2.38	2.82	136.04	195.22	2.13	-	382.77	-	24.83	13.14	-	-
	5	2.09	3.17	130.08	172.01	2.21	1.49	82.69	92.05	5.30	-	708.57	-	3.55	4.40	-	-

7 Comments on the Object Oriented Terrain Database And the Editor

This chapter will focus on the overall significance of the OOTDB and its editor. Comments will be made about design issues and the use of alternative data structures. The strengths and weaknesses of the terrain database and the graphical editor will be discussed. A discussion of unresolved issues concerning the OOTDB and editor is given. Finally, the future of this new technology is explored.

7.1 Accomplishments and Lessons Learned

The OOTDB was a proof-of-concept experiment. Because there were many unknown variables at the start of the project, some of the results obtained were unpredictable. In this section, some specific conclusions and recommendations will be presented about the object oriented design, the terrain editor and the use of Smalltalk-80 and GemStone.

7.1.1 Object Oriented Representation of Terrain Geometry

One of the main ideas presented in this project was the use of object oriented technology as a foundation for the construction of the terrain database. Object oriented design was chosen because it facilitated the natural representation of real world data in the form of structured programming.

The decomposition of a region of the world can be described spatially by using the part-whole relationship. It is easy to say that a mountain range has a city and lake located in its area. This straightforward approach to the design of the part-whole hierarchy also is used in the design of the class hierarchy. It is very natural to say that a region of terrain has water formations and man-made formations as its parts. This breakdown of the terrain into its object oriented representation is usually expressed without any difficulty, but an experienced geographer would perhaps be better equipped than a computer scientist to describe realistic terrain attributes.

7.1.1.1 Integration into the Virtual Reality Testbed

A motivating factor for the use of object oriented design is the integration of the OOTDB into a project called the **Virtual Reality Testbed (VRT)**. This project is focused on designing a standard communication protocol for simulations. It uses **players** and **ghosts** to represent dynamic models in a distributed simulation. The protocol determines how messages are sent to players which are located on remote nodes [Hughes 90].

The OOTDB will be one player located on the node which contains the permanent database. A simulation player will be able to query the terrain without knowledge of its physical location on the network. The smooth integration capability of the OOTDB into the VRT has implications for simulations which are designed using the VRT. By using the terrain database as a player, access to the complicated spatial structures, query processing power and real world data will be seamless.

7.1.1.2 Complex Queries

Incorporated into this project are methodologies from object oriented databases. It is important to realize that the terrain database is not an implementation of a traditional, query processing data manager. Rather it is a specific collection of classes which can manipulate dynamic terrain databases efficiently.

The query language processor of the terrain database is a part of the Smalltalk-80 language. The database modeler must specify the operational code of the query, and the host system will do the dynamic binding of the query call. This implementation of an object oriented database will support the complex queries which are generated from a dynamic terrain simulation. Queries such as those from a GIS and a real-time simulator can be supported by the OOTDB. The work of this project has focused on the efficient representation of spatial data. Therefore more work will be needed to fully demonstrate the real-time capabilities of the OOTDB.

7.1.1.3 Different Spatial Data Structures

Object oriented design states that the problem statement should be broken down into the base structures to determine the relationships which exist among different the entities. Once these relationships have been established, the common structures are found and placed in classes. These classes become the abstract classes, placed near the root of the class hierarchy. The class hierarchy is then refined and tested, and the design process is repeated until the resulting hierarchy is satisfactory. Using this method, the terrain database was formed for this project. It contains one root class which manages all of the part-whole hierarchy and lets more specific subclasses use Terrain's resources.

After looking at the design process used for this project, it became clear that some of the original design could have been done more carefully. This is evident in the spatial data management. If a structure like Antony's [Antony 88, 90] was followed, the performance and modularity might have improved. Specifically, a better merger of the class and part-whole hierarchies and spatial relationships would have resulted in better data management. Some design considerations, like the one mentioned above, were discovered late in the implementation of the project. It also became clear that the DCEL structure was a good choice for the spatial relationship, but the quad-tree might have been easier to work with, and most likely, a combination of these two data structures would have been very useful.

7.1.1.4 Frameworks

One final comment on the design of the terrain comes into light when looking at object oriented design. One of the most important aspects of object oriented design is the natural tendancy to reuse classes. A particular form of reuse is called a **framework** [Johnson 88]. This would give the class structure of the database as much reuseability and modularity as possible. It would also be convenient when distributing the terrain database to users. They would receive a copy of the terrain framework and would only need to know its protocol to be able to use it.

7.1.2 The OOTDBE

The OOTDBE was developed to provide a convenient interface to the OOTDB. It serves as a test platform in which terrain databases can be built and experiments can be conducted. Although it was not meant to be a production database modeling tool, it does serve a specific need in this project. There are some advantages and disadvantages in using the OOTDBE .

The user interface of the editor is an interactive graphic environment. Because of the Smalltalk-80 base language, all of the windowing and graphics routines are supplied. With the editor, the user can construct non-trivial terrain databases by interactively manipulating the part-whole hierarchy. Once a database has been created and stored in GemStone, other workstations have access and may perform operations. Additionally, details of the terrain are easy to distinguish when using a graphical environment.

An advantage of using Smalltalk-80 as the host environment is the easy integration of GemStone. The OOTDBE is the link between the OOTDB and permanent storage GemStone system. It provides a way for the user to specify which databases are stored as part of the permanent database collection.

One disadvantage when using the OOTDBE is creating the terrain classes. Smalltalk-80 provides an excellent Class Browser which gives the user the ability to create, edit and delete classes. The OOTDBE does not have a Terrain Class Browser and creating classes requires the use of the Smalltalk-80 tool.

7.1.3 Smalltalk-80

The Smalltalk-80 programming environment was chosen because it was the only available application which met the original design goals. It is available on popular UNIX workstations and is relatively inexpensive. But the use of this tools had some effect on the outcome of the experimental results in terms of performance.

One capability of the database is the ability to manipulate large amounts of data. The Smalltalk-80 system is image based, meaning that all source code is stored in binary format in the environment. The image also contains objects and its size is determined by the number of objects currently in use. As the object count increases, the internal memory manager allocates space. Large terrain databases may be slow in processing because of the virtual image, but they are only limited in size by the image size.

The only implementation of the database is in Smalltalk-80. This programming environment is available on very fast UNIX workstations and its performance on those computers is quite good, for some applications. An advantage to using Smalltalk-80 is that the language supports late binding, which means that classes can be added while an application is executing. This is a powerful language mechanism which is available in Smalltalk-80, interpreted rather than compiled language. This price of using an interpreter is performance.

Another disadvantage is Smalltalk-80's lack of popularity among computer and simulation professionals. Because object oriented ideas are new, many commercial manufacturers have not accepted this as a legitimate technology. Implementing this project in Smalltalk-80 meant that many people would be exempt from using it and rehosting would require a large effort. Recent releases of Smalltalk-80 have given this product more exposure to computer professionals. In the future, object oriented programming will become popular in all aspects of computing. Mandates by the government in the from of software standards will give this technology more exposure.

7.1.4 Gemstone

One of the other strengths as well as weaknesses is the software package GemStone Object Oriented Database. The most useful part of this tool is the persistence of Smalltalk-80 objects. It is very easy to use GemStone to permanently store and retrieve objects. Using GemStone in its basic form requires very few commands, and the objects can transfer back and forth between Smalltalk-80 and GemStone easily.

One of the main problems in using GemStone is access time. Even though the workstation which houses the GemStone server has a high processor speed, database access is always slow. Typically, the time to read a small (four patch) terrain database into Smalltalk-80 is about 30 to 60 seconds. This speed would be unacceptable for any real-time simulation.

Another drawback is moving classes from Smalltalk-80 to GemStone. The only way to transfer class definitions to GemStone is by hand. Each Terrain class and its subclasses must be meticulously recreated using the GemStone Class Browser. There is no automatic of doing this transfer.

7.2 Additional Research Needed

This section of the chapter is devoted to exploring additional work which needs to be done to better understand the usefulness of the OOTDB.

Performance analysis, real-time implementations and future directions for the OOTDB will be discussed.

7.2.1 Performance Analysis

One important aspect of this project was that the design of the database and the spatial management system would be computationally efficient. The results of the experiment presented in chapter 6 did not cover any analytical study which would have determined order statistics for database algorithms. It only found that the use of GemStone as a real-time object oriented database system is not possible. An analytical study as well as experiments to verify the theoretical orders are needed.

7.2.2 Real-time Implementation

It is evident that the Smalltalk-80 and GemStone implementation presented in this report would not be suitable for a real-time simulation because the query processing is not fast enough. To be able to achieve real-time performance, a rehosting of the database in the C++ language may be necessary. There are some disadvantages is using C++ as a basis for the terrain database. C++ does not support a dynamic class hierarchy, which prevents any reorganization of the base class structure while the database is active. A change to a class must be done off-line and the database source code must be recompiled. An advantage to using C++ would obviously be the speed.

Another improvement which could be made to the terrain database is an analysis of the innermost loops and traversals of the database data structures. There may be some places where improvements can be made to the flow of code without changing its functionality. Once a final version of the database implementation is in place, it would then be interesting to look at the possibility of designing a database machine. This would be a hardware implementation of the terrain database, and it would be done to improve performance.

7.2.3 Future Directions For OOTDB

One important area of discussion that was brought out in this report was the use of object oriented designs and databases. The future of these technologies has impacts on the next generations of the OOTDB. But what is on the horizon for these products?

7.2.3.1 Distributed Processing

The main consensus is that object oriented research will be investigating distributed processing. It was evident at the OOPSLA'90 conference that concurrent and parallel implementations of object oriented products would be the next generation of research topics. Nine out of thirty two papers and panels were devoted to either concurrent objects or parallel object oriented programming [OOPSLA 90]. It is obviously an area of active research and will be so in the future.

To continue with the topic of distributed object oriented programming is the notion of a distributed OOTDB. It would be interesting to investigate how the terrain database can "live" on separate nodes while participating in one simulation. The class and part-whole hierarchies may be located on different nodes and the interaction of the simulation players on those nodes will determine where the different parts of the database will live. This line of thinking is in concert with the distributed and parallel research in object oriented programming above. Not only does this have enormous implications in the networking of simulators, but the implementation of a distributed terrain database may mean that thousands of players may be able to participate in a single virtual world.

There are many considerations when looking at distributed databases. One of the main questions asked is how to distribute the terrain data throughout the nodes of the simulation. Some objects may need to reside on more than one node. If a part of the terrain database needs to be used on one more than one node at a time, there is a question as to which of those places would actually get the terrain object, and which would have ghosts.

Another question to be answered would be how to establish communication between objects on different nodes. This raises questions in message route planning and the configuration of an underlying physical network.

A final area of investigation would be in the area of a distributed terrain class hierarchy. Would the entire class hierarchy be replicated on each node, or would some encapsulated version of the class hierarchy be distributed? These questions, as well as many that were not mentioned here will be the focus of work on the OOTDB in the future.

7.2.3.2 Effectiveness as a Training Simulation

Also worth mentioning are possible human experiments using the OOTDB. One of the important factors of using a real-time simulation is its ability to effectively train. Many times human factor experiments are used to determine how well subjects perform while using new equipment and software. This type of analysis is needed to determine how effective the OOTDB is in a simulation environment.

7.2.3.3 Physical Modeling

Another important aspect of this work is the connection of the database with other simulator components. There needs to be research in the areas of physical modeling, soil dynamics and dynamic terrain. These aspects of simulation along with a terrain database will improve the usefulness of a simulator. Physical and constraint modeling will give the database symbolic resolution of constraints which may be placed on terrain objects. Soil dynamics will determine exactly how the terrain will react to its environment. Dynamic terrain will lead to algorithms and data structures which will better represent the terrain as it changes. All of the simulator activities mentioned above should be supported by the OOTDB.

Bibliography

[Antony 90] Antony, R. "A Hybrid Spatial / Object Oriented DBMS to Support Automated Spatial, Hierarchical and Temporal Reasoning," Draft: Advances in Spatial Reasoning. Ablex Press, NJ, 1990.

[Antony 88] Antony, R. "Representation Issues in the Design of a Spatial Database Management System," United States Army Symposium on Artificial Intelligence for Exploration of the Battlefield Environment, El Paso, TX, November, 1988.

[Atwood 90] Atwood, T., "Perspectives on Object-Oriented Databases," Hotline on Object-Oriented Technology, SIGS Publications, New York, NY, 1990.

[Banerjee 88] Banerjee, J., Kim, W., Kim, K. "Queries in Object-Oriented Databases," IEEE, 1988.

[Coad 90] Coad, P., Yourdon, E. Object Oriented Analysis. Prentice-Hall Inc. 1990.

[Deux 89] Deux, O., "The Story of O2," Altair, Le Chesnay Cedex, France, 1989.

[Egenhofer 88b] Egenhofer, M. J., Frank, A. U. "Object-Oriented Databases: Database Requirements for GIS," Department of Surveying Engineering, University of Maine, Orono ME, 1988.

[Egenhofer 88a] Egenhofer, M. J., Frank, A. U. "PANDA: An Extensible DBMS Supporting Object-Oriented Software Techniques," Department of Surveying Engineering, University of Maine, Orono ME, 1988.

[ESIG 89] "ESIG-LC Modeling System Manual," Evans and Sutherland Simulation Division, Salt Lake City, UT, 1989.

[Gargantini 82] Gargantini, I. "An Efficient Way to Represent Quadtrees," Communications of the ACM, pp. 905-910, December 1982.

[GemStone 90] Servio Logic Corp., GemStone, Alameda, CA, 1990.

[Glassner 90] Glassner, A. Graphics Gems. Academic Press Incorporated, 1990.

[Glassner 89] Glassner, A. An Introduction To Ray Tracing. Academic Press Limited, 1989.

[Goldberg 89] Goldberg, A., Robson, D. Smalltalk-80: The Language. Addison-Wesley Publishing Company, 1989.

[Hendly 90] Hendly, G, "A Class Hierarchy Tool for an Object Oriented Programming Environment," Masters Project Report, Department of Computer Science, University of Central Florida, Orlando, FL, 1990.

[Hughes 90] Hughes, C. E., Blau, B., Li, X., Moshell, J. M. "The Virtual Reality Testbed Smalltalk Prototype," Institute For Simulation and Training, Visual Systems Laboratory Memo 90.14, October 1990.

[TIGRIS 90] Intergraph, TIGRIS, Huntsville, AL, 1989.

[ITASCA 90] ITASCA Systems, Inc., ITASCA, 1990.

[Johnson 88] Johnson, R. E., Foote, B. "Designing Reusable Classes," Journal of Object Oriented Programming, pp. 22-35, June/July 1988.

[Karimi 90] Karimi, S., Personal Communications, October 1990.

[Kim 90] Kim, W. "Architectural Issues in Object-Oriented Databases," Journal of Object Oriented Programming, pp. 29-38, March/April 1990.

[Meyer 88] Meyer, B. Object-oriented Software Construction. Prentice Hall International, 1988.

[Moshell 90] Moshell, J. M., Goldiez, B., Blau, B., Dunn-Roberts, R., Klasky, R., Lisle, C., Morie, J. "A State of the Art Report on Visual Simulation," Institute for Simulation and Training, Visual Systems Laboratory, 1990.

[Objectivity 90] Objectivity, Inc., Objectivity, Menlo Park, CA., 1990.

[OOPSLA 90] Conference on Object Oriented Programming: Systems, Languages and Applications and European Conference on Object-Oriented Programming, Ottawa, Canada, October 1990.

[Orenstein 86] Orenstein, J. A. "Spatial Query Processing in an Object-Oriented Database System," Proceedings of ACM SIGMOD'86 International Conference on Management of Data, Washington D.C., pp. 326-336, May 1986.

[Orenstein 88] Orenstein, J. A., Manola, F. A. "PROBE Spatial Data Modeling and Query Processing in an Image Database Application," IEE Transactions on Software Engineering, 1988.

[Riet 90] van deRiet, R. P. "Impressions of the First International Conference on Deductive and Object Oriented Databases," Journal of Object Oriented Programming, pp. 45-48, March/April 1990.

[Stanzione 89] Stanzione, T. "Terrain Reasoning in the SIMNET Semi-Automated Forces System,." Symposium on Geographical Information Systems for Command and Control 1989, The Hague, Netherlands, October 1989.

[Versant 90] Versant Object Technology, Versant, Menlo Park, CA, 1990.

[Wegner 90] Wegner, P., "Concepts and Paradigms of Object Oriented Programming," SIGPLAN OOPS Messenger, August 1990.

[Wegner 87] Wegner, P. "Dimensions Of Object-Based Language Design," Proceedings of ACM's OOPSLA'87, pp. 168-182, 1987.