Bolt Beranek and Newman Inc.

(12)

AD-A134662

Report No. 5419

Large-Scale Simulation Network Design Study

October 1983

Prepared for:
Defense Advanced Research Projects Agency

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO. AD-A134662	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Large Scale Simulation Network Design Study		5. TYPE OF REPORT & PERIOD COVERED Special Technical Report 4/1/83 to 9/30/83
		6. PERFORMING ORG. REPORT NUMBER 5419
7. AUTHOR(s) R. Gurwitz, E. Burke, J. Calvin, A. Chatterjee, M. Harris, R. Koolish, D. Miller, and P. Yoo		8. CONTRACT OR GRANT NUMBER(s) MDA903-83-C-0168
9. PERFORMING ORGANIZATION NAME AND ADDRESS Bolt Beranek and Newman Inc. 10 Moulton Street Cambridge, MA 02238		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS ARPA Order No. 4739
11. CONTROLLING OFFICE NAME AND ADDRESS Defense Advanced Research Projects Agency 1400 Wilson Blvd. Arlington, VA 22209		12. REPORT DATE October 1983
		13. NUMBER OF PAGES 124
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) DSSW Room ID-245 The Pentagon Washington, DC 20310		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

APPROVED FOR PUBLIC RELEASE/DISTRIBUTION UNLIMITED

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Computer simulation, computer networks, computer graphics, large scale gaming (LSG), generic team training technology

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

The Large Scale Simulation Network (SimNet) project is an attempt to develop distributed, multi-player gaming technology centered around a simulation of the M1 Abrams Tank. It differs from traditional computer-oriented approaches to training simulators in several important ways. First, it seeks to develop a multi-player gaming environment. Users of the system will play against each other in simulated M1 battle tanks, rather than against a computer-driven opponent. Second, the players will

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

20.

not necessarily be required to be physically co-located. Rather, by use of computer networking, groups of players may be geographically distributed over long distances (perhaps thousands of miles), yet will be able to interact in the game as if they were in close proximity. Third, the environment will be extensible to include many players involved in a single excercise. Ultimately, the system will be able to be used by hundreds of individual players. Finally, it attempts to portray a realistic environment in which players may operate with much of the functionality and constraints of the M1 tank. Unlike traditional trainers, SimNet will offer the player the freedom to operate as he would in a real tank, succeeding or failing in assinged missions as he or she would in a physical battlefield exercise.

This report describes the design of a prototype implementation of the SimNet training system. Part I gives a broad overview of the scope and purpose of the system, as well as the principles and assumptions that have influenced the design. Part II is a detailed description of what the system will do, and the functions of its various hardware and software components. It includes a sample scenario of the system in operation. Part III is a detailed technical description of the hardware and software architecture of the system. Part IV outlines future directions for SimNet, including possible enhancements of the prototype, scaling of the approach to large groups of simulators, and the application of generic concepts of team training developed in the project to other domains.

Report No. 5419

Large Scale Simulation Network Design Study

R. Gurwitz, E. Burke, J. Calvin, A. Chatterjee,
M. Harris, R. Koolish, D. Miller, P. Yoo

October 1983

This research was supported by the Defense Advanced Research Projects Agency under Contract No. MDA903-83-C-0168, ARPA Order No. 4739, issued by Department of Army, Defense Supply Service-Washington, Washington, DC 20310.

Submitted to:

Director
Defense Advanced Research Projects Agency
1400 Wilson Boulevard
Arlington, VA 22209

Attention: Program Management

Table of Contents

FIGURES

PART I INTRODUCTION

1 Introduction

1.1 Overview of the Project

The Large Scale Simulation Network (SimNet) project is an attempt to develop distributed, multi-player gaming technology centered around a simulation of the M1 Abrams Tank. It differs from traditional computer-oriented approaches to training simulators in several important ways. First, it seeks to develop a multi-player gaming environment. Users of the system will play against each other in simulated M1 battle tanks, rather than against a computer-driven opponent. Second, the players will not necessarily be required to be physically co-located. Rather, by use of computer networking, groups of players may be geographically distributed over long distances (perhaps thousands of miles), yet will be able to interact in the game as if they were in close proximity. Third, the environment will be extensible to include many players involved in a single exercise. Ultimately, the system will be able to be used by hundreds of individual players. Finally, it attempts to portray a realistic environment in which players may operate with much of the functionality and constraints of the M1 tank. As far as possible, the simulated tanks and playing environment will offer only the constraints of the "real world," rather than imposing artificial "rules." Unlike traditional trainers, the SimNet system will offer the player the freedom to operate as he would in a real tank, succeeding or failing in assigned missions as he would in a physical battlefield exercise.

The context in which the simulation takes place is a patch of computer-modeled terrain, over which the players can travel in their simulated vehicles. Each vehicle will have a crew of four players (commander, gunner, driver, loader) who have one or more computer-generated color graphics displays and a set of controls modeled after those in the actual M1. Each of the player's displays shows a perspective view of the outside terrain, including terrain features such as hills, roads, trees, streams, etc., opposing tanks passing within the field of view, effects of weapons fire, and supporting vehicles (M2 and M3 Bradley Fighting vehicles, fuel and weapons resupply locations, etc.) Voice communications within the M1 (intercom) and between tanks and command and support elements (radio) are provided. Operating as a team, the crew of each vehicle can move over the terrain, shoot, and communicate with other tanks. The dynamics of the vehicles are simulated, as are the effects of weapons fire, the constraints of the terrain and obstacles on it (hills, rivers, trees, etc.), and maintenance characteristics of the vehicle (MTBF, MTTR). Within this environment, individual M1 crews, as well as larger units, can conduct simulated exercises to train strategy and tactics, command, and group battlefield interactions.

The hardware architecture and software needed to support this system are the subject of this report. Bolt Beranek and Newman Inc. (BBN) has been tasked to design and implement all computer hardware and software in the SimNet system. We have divided this task into simulation, graphics, and networking components. High-level design specifications for each of these components will be described. In addition, the overall architecture for the system will be described, as will the functionality of the resulting system. We will concentrate here on technical details of the SimNet system, rather than training philosophy, how the system should be used, or other factors that do not relate to the hardware and software architecture. These will be discussed in a companion report by our co-contractor, Perceptronics Inc.

The work described here took place in the period 1 April to 30 September 1983. This study phase of the project centered on overall architectural design, familiarization with the M1 tank, functional design, hardware selection, and detailed hardware and software design in each of the three implementation areas. While this report summarizes our progress in this period, it should not be considered final or complete. Rather, it should serve as a guide to the development of various phases of SimNet. It is intended as a description of the basic concepts embodied in the initial SimNet design.

1.2 Overview of the Document

This report speaks to several audiences. It is intended for the program managers and advisors, the potential military user community, and our co-contractors. Different readers will be interested in different aspects of our design and thus different sections of this report. Recognizing this fact, we have attempted to divide the material into sections that contain topics of interest to various groups of readers. The introductory sections of Part I give a broad overview of the scope and purpose of the system, as well as the overall principles and assumptions that have influenced the design. Part II is a detailed functional description of what the system will do, and the functions of its various hardware and software components. It includes a sample scenario of the system in operation. Part III goes into technical detail on the hardware and software architecture of the three major engineering areas in the system: graphics, networking, and simulation. Part IV is the conclusion, which discusses future expansion of the system, scaling to larger groups of simulators, and the application of the generic team training ideas developed in the project to other domains. Following that, an appendix contains additional material on the design, including a detailed game session scenario.

2 Philosophical Considerations

The intent of the Large Scale Simulation Network project is to demonstrate an important new training technology. We are not, at this stage, attempting to develop a complete training system -- a far more ambitious goal -- and it is important to keep this distinction clearly in mind.

Since we are not now attempting to construct a training system, we will not devote substantial amounts of time to hypothetical discussions of how such a system might ultimately be used. We should, however, attempt to insure that our demonstration system can be gracefully extended into a complete training system, if it cannot, the demonstration may be rejected or ignored by the community of potential users.

Once the feasibility of the technology has been established, it will then have to be evaluated in the context of a specific training system to determine its effectiveness and the ways it can best be used. While it is obviously too early to make any definitive statements, we expect the simulation network to be used within a framework of command and communication that is very similar to the one currently used in field maneuvers and war games. The normal communications paths among platoons, companies, and battalions would be employed in essentially the same way that they would be used if the simulated tanks were real. In fact, one can easily imagine the command and control aspects of the simulation being sufficiently realistic that an observer in a rear-echelon command post might be unaware that the vehicles involved are only electronic simulations.

2.1 Design Principles

In the course of our technical feasibility and design studies, we have sought inputs from a variety of sources, including several representatives from the military user community. As a result of these discussions, we have identified a series of design principles to be followed in the development of SimNet.

1. The terrain display presented at each vehicle simulator

must be adequate to permit visual navigation with the aid of a simplified topographic map. The ability of a tank crew to locate their position and plan their actions by means of such a map is a fundamental skill that has profound implications for the success of any tactical operation. If the SimNet design does not reflect this fact, its utility as a training aid will be greatly (perhaps fatally) diminished.

2. The visual displays in the simulated vehicles must provide a sufficient visual field of view to permit the coordination of tactical maneuvers involving several vehicles. In particular, this principle implies that the displays for the driver and the commander should cover a horizontal angle of approximately 180 degrees, so that several tanks can advance "on line" while maintaining visual contact.

3. The SimNet software will represent a passive enforcer of the "rules of the game." No attempt will be made to simulate the decision making of enemy forces. The outcome of events will thus be determined through the interaction of human initiatives and models of physical laws and constraints.

4. The SimNet software must preclude actions that would appear obviously unrealistic to the participants. While it is not necessary to provide realistic models of every possible physical effect, it is important to maintain an adequate atmosphere of realism. This atmosphere can be shattered if the simulation software permits situations to occur that would be impossible in the real world.

5. The SimNet software must prevent the use of tactical "tricks" that would not be possible on a real battlefield. In addition to damaging the illusion of reality, the availability of such unrealistic maneuvers might lead participants to become dependent on them, producing counterproductive training effects.

6. Similarly, the simulation should not provide the participants with additional sources of training information that would not be available on a real battlefield. To the extent that they learn to depend on

such information, counterproductive training effects would also occur. In particular, no commander should be able to obtain an "omniscient" view of the battlefield -- he must make his decisions with the same kinds of fuzzy and incomplete information that he would have to use on a real battlefield. (It should be pointed out, however, that nothing in the system design we are developing would preclude capturing such information from the network for subsequent replay and analysis.)

7. Usage, resupply, and reconstitution of expendable supplies and equipment should be constrained to reflect battlefield realities. Nothing important should be "free," and running out of supplies should entail consequences that approximate those of the real world. For example, a tank that runs out of ammunition will have to drive to an ammunition dump. A tank that runs out of fuel will have to wait in place for a time interval that realistically reflects the time required for a fuel truck to reach the tank's current location.

2.2 Effects to be Included in the Simulation

In our meetings regarding the Large Scale Simulation Network, the design principles just enumerated have been discussed extensively, along with their implications for specific effects to be modeled and included in the simulation. A reasonable consensus has evolved that certain areas are especially critical to an effective simulation.

The terrain is one of the most important. As noted in the previous section, the terrain model must supply sufficient detail to permit the crews to locate their position and navigate with the aid of a simplified topographical map. This requirement implies that there must be an adequate number of recognizable landmarks (hills, ridges, roads, rivers, tree lines, silos, barns, etc.) in each terrain patch. It is also crucial that the terrain provide enough smaller features (ditches, boulders, etc.) to permit the tanks to adopt appropriate defilade positions when necessary.

Vehicle dynamics must be simulated with enough fidelity that none of the vehicles appear to do unrealistic things. Engine speed, vehicle speed, hill-climbing ability, turret slew rates, etc., must be reflected appropriately in the dynamic equations used for the simulation.

Ballistics must also be modeled to a level of detail at which no obviously unrealistic effects are observed. This means that the ballistic characteristics of different types of ordnance (HEAT, HEP, Sabot, etc.) must be depicted with reasonable accuracy, and that ranges and kill probabilities must also be realistic.

Logistic considerations must certainly include ammunition and fuel. Learning to allocate these resources wisely is a crucial element of crew training, and is a principal component of the supply and distribution functions of Combat Service Support. As noted in the previous section, refueling and resupply must consume realistic amounts of time and be conducted only at appropriate locations (such as fuel and ammo dumps).

Maintenance considerations must be incorporated to the level necessary to reflect the approximate rate of various types of failures (mobility, communications, weapons) under combat conditions. A training system that ignored these considerations might encourage commanders to adopt strategies and tactics that ignore the effects of such failures, and that might therefore be disastrous if applied in combat.

Radio communications must be incorporated, since this is the primary mechanism for command and control above the platoon level. An appropriate number of channels must be provided for various types of communications

Artillery effects will be included as well, to reflect the communications and coordination required to make effective use of friendly artillery as well as to reflect the threat posed by enemy artillery.

2.3 Effects to be Excluded from the Simulation

In the course of the SimNet project meetings, it has also been agreed that certain effects will be excluded from the simulation, at least for the present. In some cases, the effects in question are clearly peripheral to the main training purposes of the system, in other cases, the effects would involve adding substantial complexities to the modeling task that do not seem cost effective at the present time

Night fighting, for example, would involve an entirely different set of analyses and design decisions for the graphics components of the simulation.

The use of smoke would require additional levels of graphics processing This may be feasible, however, and will be considered for inclusion in later stages of the project.

Electronic countermeasures would involve additional controls and mockups of displays. These effects could also be considered for inclusion in later stages, if they are judged to be sufficiently important to merit the added cost and complexity they would involve.

CBW and nuclear warfare has been excluded on similar grounds of cost and complexity.

Mines would require additional complexities in the terrain data base, especially if they could be laid down in the course of the game. In addition, mine detection and clearing procedures would be hard to simulate realistically.

Changes in terrain features, such as blowing up bridges, have been excluded from the current simulation, but represent a candidate for early inclusion in later stages.

Infantry and similar combat elements would be very difficult to incorporate into the simulation at the level of individual troops. However, it might be possible to incorporate some representation of squads or platoons of infantry, controlled off-line by a commander, into later stages of the project, using appropriate icons.

The M1's machine guns are being excluded from the present stage of the project because of the exclusion of infantry elements, and because these guns are ineffective against heavy armor.

The incorporation of medical and mess services, and the like, seems to add little to the simulation. These elements are not likely to be incorporated at any stage of the project.

PART II. FUNCTIONAL DESCRIPTION

We now begin a functional description of SimNet from the point of view of the participant in a SimNet exercise. In addition, we provide an overview of the various components that comprise the system and how they are interconnected. The purpose of this part of the report is to orient the reader to the system and its components, and to define terminology used in the remainder of the document. It also serves to describe how the system will appear to the user, gives the "rules of the game," and tries to give the reader a feel for how the system operates. To aid in visualization, a brief game scenario is included in this part. A more detailed sample session can be found in the appendix.

Part III will discuss the implementation of this model in technical detail. Readers who are interested only in function may choose to confine themselves to Part II of the report.

3 Functional Overview

3.1 Introduction

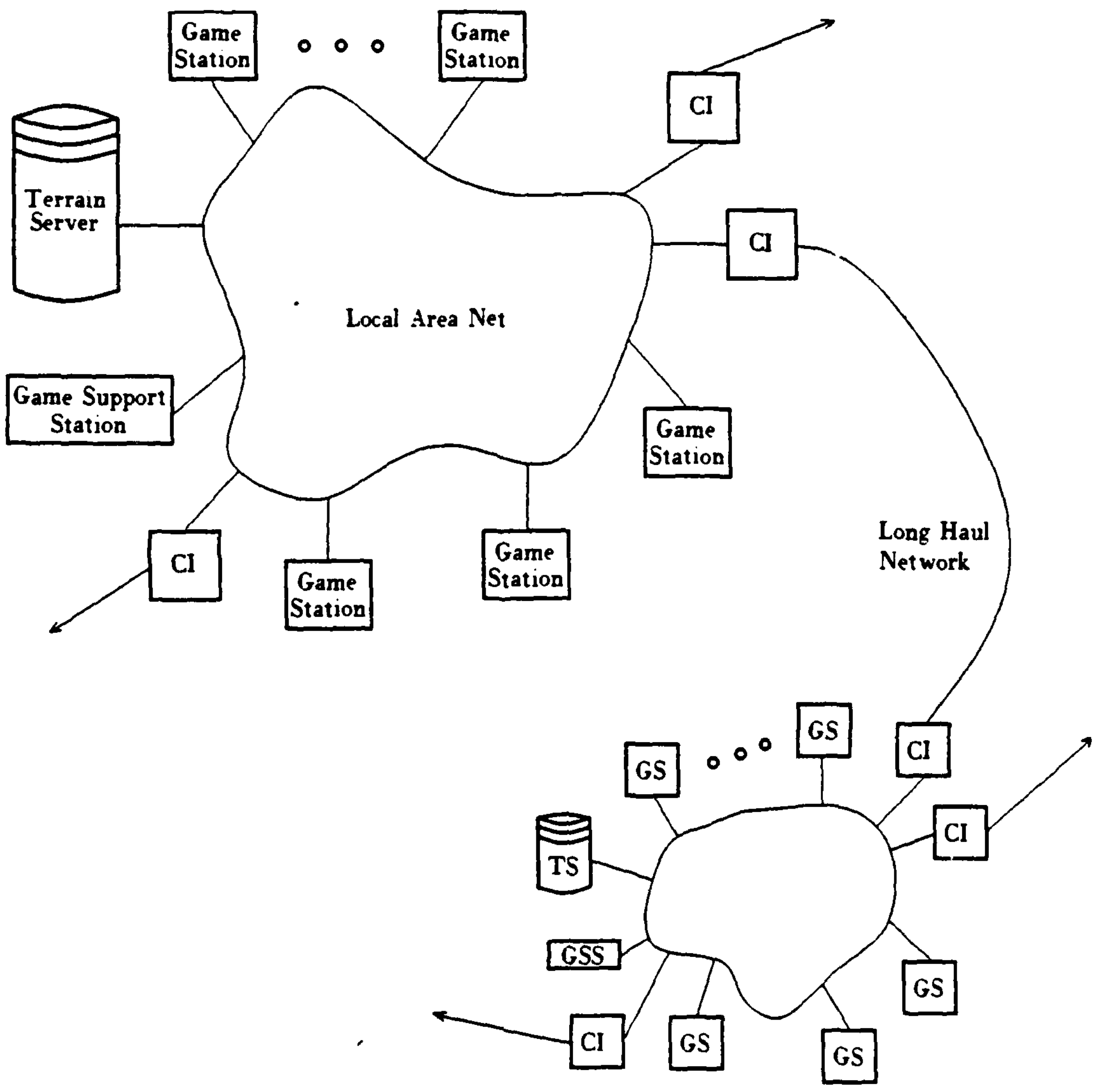
The SimNet system consists of simulated vehicles manned by crews that fill the various crew positions in the vehicle. The first vehicle to be simulated will be the M1 Abrams tank, with a crew of four. The crew is housed in an enclosure designed to resemble the interior of the M1 tank. Each crew member has one or more color video displays through which he can see a perspective view of a three-dimensional terrain patch on which the battle exercise takes place. The crew also has interaction controls that allow them to drive the vehicle over the terrain and to aim and fire weapons. The crew can engage the other vehicles in the simulation as they are encountered in the terrain. The object of the system is to present a simulated battlefield that has many of the characteristics of a real battle situation.

Each of the vehicles is simulated by a Game Station (GS), which consists of processors, displays, and interaction controls. Several GSs are connected together on a Local Area Network (LAN). A Game Support Station (GSS), also connected to the LAN, consists of a processor and disk storage. It is used for exercise support and control functions and is manned by a Gamesmaster (GM), whose function is overall control of the sy .em. Besides the GSS, a Terrain Server (TS) is connected to the LAN. A TS consists of a processor and disk storage used for distributing the terrain graphics database to the GS, and TS and GSS may both share the same processor and disk. The stations on the LAN are connected to other networks of stations by a Long Haul Network (LHN) and Communications Interface processors (CI). The interconnection of these components is shown in Figure 1. They are described in detail in Part II.

3.2 Terrain

The terrain seen through the view ports of the simulator console will be rather stylized and includes navigational features such as hills, rivers, roads, tree lines, etc. The ground plane has "painted" on it flat polygonal objects such as roads, rivers, and ponds. Hills are objects of spherical shape, and do not intersect any of the polygonal objects mentioned previously. They are of such slope as to be navigable by tanks. Additionally, the terrain contains trees of various heights, bushes, groves, and boulders. Tanks are able to drive through bushes but are blocked by the other objects. Finally, the ground in the immediate vicinity of a tank has texture points scattered randomly across it, which aid in depth perception of the terrain.

The color of the background varies depending on the distance. the ground nearest the vehicle looking bright green, suggesting grass, towards the horizon the ground color changes from bright green to a dull green, and finally to a bluish grey at the horizon. The color of the sky is bluish white at the horizon, changing to a bright blue at higher angular elevations. Trees have green tops and brown trunks. Lakes and rivers are blue, boulders are brown.



SimNet Components
Figure 1

3.3 Vehicles

Gun elevation and turret rotation are visible to nearby tanks. Farther away, the vehicles are represented by icons that have varying degrees of gun elevation or turret rotation. In both instances, the color of the tanks is olive green.

Tanks can fire several types of rounds from the main gun (HEAT, Sabot, etc.). Depending upon the type of ordnance used, an appropriate muzzle flash will be seen from other vehicles in the vicinity of the firing tank. The crew of the firing tank will also see a flash effect. When a tank is hit, its crew will see a bright flash followed by darkness and hear the sound of an explosion. The tank that fired the round will see hit effects based on the damage sustained by the target tank.

3.4 Descriptions of Crew Positions and Functions

An M1 tank is operated by a crew of four: the <u>tank commander</u>, <u>gunner</u>, <u>driver</u>, and <u>loader</u>. The relative positions of the tank crew can best be described in terms of a plan view of the tank. The driver is forward, in the hull of the tank, in a semi-supine position; behind him and to the right is the gunner's seat; behind the driver and to the left is the loader's seat, behind and above the gunner is the tank commander's seat. The gunner's, tank commander's, and loader's seats are in the turret, which rotates with the main gun. Above the tank commander's and loader's seats are hatches that open, allowing them to look outside from a standing position. When the hatches are closed, the tank crew can see outside by means of periscopes with limited field of view. In the SimNet GS, these periscopes will be simulated by color video displays: three for the tank commander, three for the driver, one for the loader, and one for the gunner. The solid angles subtended by these displays at the respective crew positions will approximate the angles presented by the appropriate periscopes in the actual M1 tank. The simulation portrays operation of the tank in the "closed hatch" position.

Another point that needs special mention here concerns the loader. In the M1 tank, the loader accesses rounds of various types by activating a knee switch to operate the ready ammunition bustle door. In the SimNet GS there are two lights and a switch for each round. The lights indicate whether the round is a HEAT or Sabot round. Appropriate lights are lit at exercise initialization to indicate the type and amount of ordnance available in the vehicle the GS is simulating. The loader "pulls out a round" by pushing the switch for the round. When that happens, the light for that round is extinguished. Since the M1 can fire the main gun at a maximum of twelve rounds per minute, a minimum of five seconds must elapse between successive depressions of the rounds switches.

As the names suggest, the function of the driver is to drive the tank, that of the gunner is to acquire targets and to fire the tank's main weapons, and that of the loader is to keep the main gun loaded with the right ordnance, and to help in target acquisition. The function of the tank commander is overall command of the vehicle, which includes giving navigational instructions to the driver, directing the gunner in target acquisition, and issuing "fire" and "cease fire" orders. The tank commander also communicates by radio with other tanks and rear-echelon command elements. In some cases, the tank commander may also be a platoon or company commander.

3.5 Brief Exercise Scenario

A SimNet exercise is preceded by a preparatory phase during which the tank crew is presented with a topographical map of the simulated battlefield. As part of an exercise, the participating tank crews would receive military operations orders prepared by the command structure, as they would in a real battle situation. The following scenario is based on Section 13 (particularly page 13-40) of the Large Scale Simulation Data Package on the M1 tank prepared by Perceptronics [Bloedorn et. al., 1983].

Let us imagine that a particular mission for the platoon is to perform a "deliberate attack" on the enemy presently occupying a certain hill. The hill will therefore become the "objective" of our tank crews. In order to perform this mission, certain collective skills on the part of the tank crews will be employed. For example, the crews must know movement techniques, cover and concealment techniques, communications techniques, operation of tanks, employment of weapons, tank maintenance, etc. This particular SimNet exercise will give the tank crews opportunities to acquire these skills.

Let us further imagine that the tanks of our platoon are in a "coil" formation in the middle of the countryside; the tank platoon has already been prepared for combat operations, which means fueling and loading of armament has been done, and the tank commanders have received their operations orders.

The exercise begins with the tank crews entering the GS and occupying their respective positions. In a particular GS, the tank commander looks through the view ports and establishes his location relative to the topographical map by identifying the various terrain features. Likewise the driver, gunner, and loader familiarize themselves with the controls and dials of the GS, which, by design, are similar to those of the actual M1 tank. The consoles of the GS do not function until the GM sends a "Start" command to the GS. Upon receipt of that command the exercise begins.

The platoon leader requests indirect fire on the hill at a specified time. The individual tank commanders order their drivers to drive in the direction of the hill. The platoon crosses the line-of-departure and starts moving in a wedge formation. Because the tanks are on fields, their speeds are slower than they would be on paved roads. From inside their tanks, the crews hear engine noise proportional to the engine RPM. The tank commanders give instructions to their drivers to avoid obstacles or go over them or, in the case of bushes, to go through them. The tanks even ford a small stream that is in the way. Looking at the visual displays, the crews see foreground texture points that stream past the tank as it moves. This reinforces the crew's perception of being in a tank moving across country, and aids in their depth perception of the terrain. The loaders are scanning around them to see that there are no threats on the sides. The gunners are also alert, looking through their sights for enemy tanks.

The platoon is now nearing the objective, and it is time for the indirect artillery fire to take place in accordance with the commander's earlier request. The indirect fire makes it easier for our forces to reach the objective. The platoon commander instructs the tank commanders to hide behind the hills on the right. The tanks do so with only their turrets showing above the hill (hull defilade). The artillery strike has now begun and the barrage continues for well over fifteen minutes. The tank crew sees a number of explosions in the vicinity of the objective. Our platoon commander sees the dust plumes of an enemy tank, which is almost two kilometers away. He designates two tanks to make the assault, assigning the two remaining tanks to perform an overwatch mission to provide covering fire.

The tank commander orders the driver to advance toward target. The driver releases the parking brake on hearing the drive command and drives the tank forward. The tank commander assists the gunner in acquiring the target by moving the turret and selecting appropriate gun elevation. He says "gunner, sabot, tank." The gunner takes the range of the target by engaging the laser range finder, the fire control mode switch is set to "normal," the ordered ammunition is selected, the main gun has been selected, and the gunner's primary sight is set to 10X magnification. The gunner then lays the fire control reticle on the target aiming point. This is also confirmed by the tank commander by means of his extension of the gunner's primary sight. The gunner says "identified." The loader has meanwhile gone through the motions of arming the main gun. This includes ensuring that the "safe" light is lit, depressing the knee switch that opens the bustle door, pretending to load the appropriate round by pushing a rounds switch, insuring that the "armed" light is lit, placing turret blower in "on" position, facing ammo doors, and finally announcing "up."

The tank commander waits until he hears the gunner say "identified" and the loader say "up," and then commands "fire." When the gunner hears this command he fires, announces "on the way," and continues to track the enemy vehicle. The muzzle flash is seen in this tank as well as in the other tanks if they happened to be looking in this direction. Assuming the round hit the enemy tank, the crew of the firing tank sees hit effects depending upon the amount of damage done. The enemy tank crew sees and hears appropriate hit effects.

At this point our platoon leader directs all the tanks to fire on the objective, and the tanks rapidly move forward. The enemy is overwhelmed and the objective taken. Our tanks consolidate and reorganize on the hill and prepare for an enemy counter-attack.

In this brief scenario, we did not deal with failures of the functional components of the tanks, which the simulation permits. In the same vein, ammunition and fuel are consumed as the exercise progresses, and they must be replenished from support elements when they run low. The simulated vehicle may itself be hit by enemy fire, causing the vehicle to lose its ability to move, shoot, or communicate. On these occasions, the tank commander requests maintenance/resupply from his support elements by radio. As long as the constraints on availability of supplies and the rate at which they can be delivered are satisfied, these requests are met, and the tanks become fully operational again. A more detailed scenario is presented in Appendix A (a typical SimNet exercise).

4 The SimNet Environment: Components and their Functions

The physical environment consists of four types of components: game stations, terrain servers, game support stations, and networks. These components are described briefly here as an overview, and in more detail in later sections.

4.1 Game Station (GS)

Each vehicle consists of a simulator console, a graphics processor, a simulation processor and a network interface.

Vehicle software will primarily consist of a small real-time multitasking operating system (CMOS), Interaction Device Controller (IDC), simulation software, graphics software, terrain database, and network software.

The main functions of a tank are to <u>move</u>, <u>shoot</u>, and <u>communicate</u>. In addition, tanks require <u>maintenance and resupply</u>. Let us examine these functions in a little bit more detail.

In order to perform the <u>move</u> function, the driver's controls are first checked and the values of the settings determined. The terrain database is examined and, knowing the current location of the vehicle, the terrain slope, surface quality, etc., are obtained. The vehicle's own physical constraints such as maximum speed, amount of fuel remaining, acceleration, etc., are determined; and, subject to the fact that these constraints are satisfied, the new position of the vehicle is computed. At this time, we check to see if the move to the new location is possible. For example, it might be occupied by a boulder, or another vehicle. If so, the vehicle cannot move to the new location. If it is possible to occupy the new location, the vehicle does so and updates its location data in its local state vector, which is broadcast to other vehicles. The vehicle also updates internal parameters such as remaining fuel, etc. Finally, the vehicle determines whether it needs new terrain information from the terrain server.

The shoot function involves looking at the gunner's controls, which include controls for rotating the turret, raising or lowering the gun, locking onto a target with the fire control system, panning the turret to follow the target, selecting the right magnification in the gunner's sight, selecting the right ammunition, etc., and eventually firing rounds. Ballistic calculations are performed for each round, and an appropriate hit message is sent to the target. Muzzle flashes are seen by tanks looking in this direction. Various parameters such as gun tube wear, which affect ballistics, are also maintained.

Voice communication takes place with the outside world via radio. Contact is maintained with other tanks and the chain of command of the units participating in the simulation. For example, the platoon commander gives orders over the radio, or the tank commander may ask for refueling or resupply of ammunition, or he may declare that he has been hit and requires maintenance.

Last, the maintenance and resupply function demands that tank components fail during the simulation according to their mean-time-between-failures (MTBF). When a statistical failure occurs, some part of the vehicle stops functioning, and the tank commander asks for help according to standard practices. The tank crew must first wait for maintenance personnel and supplies to arrive or drive to staging areas for them if possible. They must then wait for an appropriate mean-time-to-repair (MTTR) or resupply. These wait times determine when the tank will be back in operation. Occasionally, supplies may never arrive because they were intercepted by the enemy.

4.1.1 Simulator Console

The simulator console consists of a working mockup of the console of the M1 tank. It contains all controls and dials of the M1 that are relevant to the simulation. A list of simulated dials and controls is shown in Appendix B. The viewports of the tank, which allow the crew to look at the outside terrain, will be simulated by eight color video displays: three for the tank commander, one for the gunner, one for the loader, and three for the driver.

4.1.2 Graphics Processor

The graphics processor consists of an Adage 3000 graphics system modified with custom components developed by BBN. The Adage system includes a backplane power supply/chassis, an MA1024 high-speed coordinate transform processor, and an AGG4 micro-coded special-purpose graphics processor for rasterization and other graphics tasks. This will be augmented by custom-designed frame buffer memory and video generators to drive eight displays at 512 x 512 x 8 bit resolution.

The graphics processor provides computer-generated images for the color video displays of the console. These images correspond to the location of the tank and the direction in which the tank hull or turret are pointed. In particular, the graphics processor shows terrain, other vehicles, effects of firing rounds, effects of being hit by a round, and indirect fire, provided these are visible in the direction in which the tank hull or turret are pointed.

4.1.3 Simulation Processor

The simulation processor consists of a custom-designed dual processor MC68000 with bulk RAM and local network interface. One of the MC68000 processors handles high-level graphics data management, the other handles the simulation and control processing.

This component performs the actual simulation functions of the tank: movement, shooting, communication, maintenance and resupply. It also works very closely with the graphics processor and provides it with all the data that it needs. In addition, it participates in data exchange with other SimNet objects via the network.

4.2 Game Support Station (GSS)

There is one GSS for the whole SimNet. A GSS requires a MC68000 processor with 512kB RAM, 80MB disk, a terminal, and a network interface. In the initial implementation the GSS will be on the same processor as the TS.

GSS software includes an operating system, a command line interpreter, loader, applications software (maintenance/resupply vehicles, indirect fire), initialization data base, and networking software. In addition, there is a tool for processing raw exercise data into exercise initialization data bases.

The GSS performs two types of functions: off-line functions and on-line functions.

4.2.1 Off-line Function

The off-line function is the preparation of the exercise file (Gfile). The Gfile contains all the information about initial conditions of the present exercise that are relevant to the simulation: location and orientation of vehicles, initialization data, etc.

4.2.2 On-line Functions

These functions are directly SimNet-related and are performed by order of the GM. They are:

- Establish exercise time,
- Select appropriate terrain map file (TMfile) and game file (Gfile), about which more will be said later,
- Order TS to use selected TMfile to load terrain patches into GSs,
- Use Gfile to load initialization data into GSs,

- o Start/Stop exercise.
- o Operate supply/maintenance vehicles, and
- o Operate indirect fire

4.2.3 Terrain Server (TS)

There is at least one TS per local net (see Section 5.1.3.1). Each TS consists of a MC68000 processor, bulk RAM, a 300 MB disk, and a network interface.

TS software consists of an operating system, networking software, and the terrain database. In addition, there is an off-line tool called the terrain tool.

The function of the TS is to provide terrain data to all the GSs. It stores terrain data for a 100km x 100km piece of terrain over which the exercise is to be held. Upon demand, the TS initially provides each GS with a 10km x 10km terrain patch, which is the subset of the terrain relevant to the vehicle at its present position. As the vehicle moves, the vehicle simulation processor in the GS asks the TS to provide new terrain data (in the form of terrain patch updates) when necessary. The vehicle simulation processor performs its own internal storage management.

The terrain tool is used in the preparation of the terrain map file (TMfile) and associated terrain map. The TMfile is a machine-readable form of the terrain where the simulation exercise will be held, and the Terrain Map is a human-readable map of the same terrain. It is given to the vehicle crew in hard copy for navigational purposes. This has been discussed further in Appendix A (A typical exercise).

4.3 Networks

There are two types of networks in SimNet. local area nets (LAN) and long haul nets (LHN). These networks are functionally identical for the purposes of this discussion, and will therefore be treated as a single entity.

Besides providing a data and control link between the objects of SimNet, the network performs the following functions for the simulation.

- o Transports global time from GSS to GSs,
- o Provides vehicle data to all GSs.
- o Provides "round-fired" message to all GSs, and
- o Provides "hit" message from firing-vehicle GS to target-vehicle GS.

A more detailed exposition of the network aspects of SimNet appears in Section 6.

PART III. TECHNICAL DESIGN

This part of the report is concerned with the detailed technical design of the hardware and software aspects of SimNet. It will be of interest to those readers who are concerned with the implementation details of the system, and is of less importance to those who are interested only in functions. The reader of this section should previously have read Section 3.1 on the SimNet environment, which provides some useful definitions.

The following sections cover the three major areas of the design: graphics, networking, and simulation. Hardware and software aspects are treated separately in each area.

5 Graphics

The graphics aspects of the SimNet design are probably the most technically demanding. Three aspects of the graphics tasks are discussed: the terrain model, the algorithms and software for driving the displays, and the hardware architecture on which it is implemented. In each area, we include data size considerations and performance parameters in enough detail to make performance estimates for the resulting graphics subsystem.

5.1 Terrain Display and Database Design

5.1.1 Terrain Database Overview

5.1.1.1 Terrain Map and Feature Templates

The terrain model is a representation of the 100km by 100km playing area. The terrain model consists of two kinds of data: a set of "templates" or patterns for generic terrain features, and a "terrain map," which defines the playing surface in terms of a surface plane on which specific terrain features are placed. There is one template for each type of terrain feature. The template defines the parameters that must be specified when an instance of a feature is created. The terrain map is essentially a list of specific terrain features, represented by their parameters. This parametric representation of terrain permits the terrain model to be much more compact than a point-by-point representation, and allows for fast computation of slopes.

The terrain map is a two-level structure. The first level is a two-dimensional array where each element is a pointer to a list of features. Each element represents a square 1km on a side. The array structure allows terrain features to be grouped by geographical location, which is necessary for effecient accessing of data in the terrain database. In the second level of the structure are the lists of features pointed to by the array elements in the first level. Each list contains all the terrain features within a specific 1km square. Feature lists have no internal ordering. Upon request, groups of lists are passed to the game stations, where individual terrain features may be examined and ordered for the purposes of the vehicle simulation and the graphics display.

Some terrain features are large enough, or may be positioned in such a way, that they lie in more than one 1km square. This is handled in two ways. Features that consist of many parts, such as roads or rivers, will be divided into separate parts, one for each terrain square, so that the parts meet properly at the edges. Single large features, such as hills, will have entries in each terrain square that they occupy, even though the defining location of such a feature is in only one square. This way, all of the features that affect a given terrain square will be included in the list of features for that square.

5.1.1.2 Terrain Editor

The full terrain database is created by using a terrain editor. The terrain is built up by creating individual features and assigning them to specific locations. Since terrain creation is very labor-intensive, we expect that the terrain will be quite schematic, that is, the overall features of an area will be specified, but minute detail will be missing. Once the terrain is created, the terrain editor can then be used to generate a simplified topographical map of the terrain area suitable for use by the tank crews

5.1.2 Game Station Graphics Database

5.1.2.1 Local Terrain Patch

Since the full terrain database is quite large, and only a small part of it is of interest to a vehicle at any given time, each game station keeps only a subset of the full terrain map in its memory. This subset is called the local terrain patch. A local terrain patch is a square of 10km on a side centered on the current position of the game station's vehicle. As the vehicle moves across the playing area, it occasionally requests from the terrain server new terrain lying in the direction of travel, and it discards map data that is now outside the patch in the backward direction. The algorithm to request and discard terrain data is arranged in such a way that there will be no thrashing if the vehicle happens to meander back and forth accross the request/discard boundary. For example, if we assume the 10km by 10km local terrain patch, if the request/discard boundary is 4km from the edge, there is a 2km by 2km region in the center of the patch that the vehicle may move around without triggering a terrain update. When the vehicle crosses the edge of this center area, a terrain update is triggered, the result of which is that the vehicle is put back in the center of a modified 10km by 10km patch. The vehicle now has to move a full kilometer before another terrain update will happen. If it just turns around and crosses the boundary, no update happens because the vehicle is now 5km, not 4km, from the edge.

The local terrain patch is organized in the same fashion as the full terrain database. The top level is a 10 by 10 array of pointers to 1km square feature lists. This organization makes it very easy to add and delete terrain, since at the top level, only pointers in the array have to be moved. At the second level, a memory management scheme will allocate and delete space for the feature lists.

The distant background is handled by having a row of squares surrounding the local terrain patch, which contain objects that are outside of the local terrain patch but are large enough to be seen. When a terrain update occurs, the terrain server will send the game station the terrain for both the new row of squares at the edge of the local terrain patch, and the new distant background. The reason for not merging this information into one set of squares is that the data in the local patch can migrate toward the center of the local patch, and the distant features would have to be removed when this happened. In either case, the number of features sent from the terrain server is the same. Including the distant features with each terrain update means that there is no problem in remembering how up-to-date they are.

5.1.2.2 Transient Objects

There are two other sets of data that, along with the local terrain patch, make up the complete local graphics database. The first is the list of transient objects, such as vehicles and weapons effects. Vehicle state information is kept in a fixed size array since the number of vehicles is constant. The network interface updates this information as vehicle position messages arrive over the network.

Other transient objects, such as weapons effects and fuel trucks, will be kept on another list. Some of these effects are very transient, and the game station will take them off the list after they are displayed for the appropriate time. Features such as fuel trucks will stay in one location until they are removed by instruction from the game support station.

5.1.2.3 Texture Points

The final part of the graphics database is the list of texture points. These are small spots on the ground that act as depth cues. One thousand texture points are generated in the 100-meter circle surrounding the vehicle. Texture points act very much like terrain features in that as the vehicle moves, new points are generated in front of the vehicle, while old points are deleted as they move out of the 100-meter circle. Texture points are generated in the game station and don't require any interaction with the terrain server.

5.1.3 Interfaces to the Game Station Graphics Database

5.1.3.1 Terrain Server

As described above, the game station has a local terrain patch that must be initialized and updated from outside the game station. The terrain server is a network server avaliable to all of the game stations on the high-speed local network. The terrain server keeps the master copy of the terrain map of the entire playing surface, and processes requests from game stations for terrain patch initializations or terrain patch updates

request	response
request_patch (station_id) (position)	The portion of the terrain map describing the patch centered on (position) is transmitted to the game station. This request is normally made once, on game station initialization.
update_patch (station_id) (old_position) (new_position)	The portion of the terrain map describing the updated patch (i.e., (new_position) patch - (old_position) patch) is transmitted to the game station.

5.1.3.2 Game Station Network Interface

Position and type information for all transient objects enters the game station through the network interface. Vehicle positions, weapons effects, and the other events are sent by other game stations or the game support station when they happen. Events within the local terrain patch are included in the game station's graphic database so that they will be properly displayed.

5.1.3.3 The Simulation

The vehicle simulation has a two-way interface to the graphics system. Variables that the simulation must provide to the graphics system are: vehicle position in X and Y, vehicle azimuth, turret azimuth, gun sight magnification and reticle position, and ballistic path of main gun shots. The graphics system provides to the simulation: Z position, ground slope, ground type, and, when a shot is fired, a list of possible targets in range order.

5.1.3.4 High-Level Graphics Processing

Features to be displayed on the video screens must be extracted from the graphics database and sent to the graphics processor. This is a two-step process. The first step is to determine which objects in the database are small enough and far away enough so that their image would be smaller than one pixel, and therefore can be eliminated from further consideration. Most of the features in the database are trees and rocks that cannot be resolved unless they are fairly close. Eliminating these objects drastically reduces the computational workload of generating the display. This selection need only be done every second or so, since the vehicle position won't change enough in that time to affect the results.

The second step of the process is to generate a list of features for each of the seven unity-power displays. To do this, each of the features selected by step one must be examined to see if it is within the field of view of any of the display windows. If it is, it is added to the feature list for that window. The selection performed by this step does not have be perfect, since the graphics processor will do the final clipping to the display window. The most important thing is to include all features that have any parts in the window. This second step will be synchronized to the display rate of 5 Hz.

The gunner's sight must be handled differently because it can work at either three or ten power. This means that the visibility check in step one must be different. For this reason, the eighth display list must be constructed from the local terrain patch every 200 ms.

5.1.4 Feature Templates

The use of templates allows terrain features to be split into two components: the individual parameters defining specific terrain features, and the common procedures used to manipulate any terrain feature derived from one template. The amount of data required to represent many given features is localized. parameters reside in the terrain map, and template procedures reside in the game stations. In addition, the composition of template procedures from carefully designed subroutines permits extensive sharing of code and further reduction in storage requirements.

5.1.4.1 Notation and Common Structures

Terrain features are defined using a three-dimensional cartesian coordinate system. The X and Y dimensions correspond to longitude and latitude, while Z corresponds to altitude. The 0,0 location is the southwest corner of the playing area. X and Y are maintained as 16-bit unsigned integers. This gives a resolution of 1.5 meters within the 100km by 100km playing area. Z is kept as a 16 bit signed integer in order to allow for terrain features such as rivers and trenches that are below the normal ground surface.

Some features have colors and textures that are implicitly defined by the type of a terrain feature. They are not explicitly given for each instance of a feature.

Features may be either simple or compound. Simple features, such as trees and rocks, have a fixed structure and can be varied only in their position and size. Compound features, such as roads or ponds, must be defined by giving a list of parameters that define the border of the feature or its linear extent.

All areas of the playing surface have a ground type that specifies the visual properties of that piece of terrain and the effect that ground type has on the simulated vehicle. Ground types are: grass, water, dirt, and road.

5 1.4.2 Passive Elements

Passive elements are fixed terrain features. Passive elements never move and cannot be created or destroyed.

Surface Plane

The surface plane is an implicit baseline for all features, and is defined at Z = 0 All features except ditches and rivers are on or above the surface plane. It has no color or texture, but is purely a measurement convention.

Simple Vertical Features

These are features such as clouds, rocks, trees, bushes, buildings, and silos. These are features for which the graphics system has a built-in representation. Some features, such as trees and rocks, will look the same from any viewpoint. Features such as buildings can be presented from different viewpoints by the graphics system.

```
struct simple_vertical_feature
    }
    int          feature_type;
    xyz_coord    position,
    byte         size,
    {,
```

Horizontal Flat Features

These are features such as ponds and fields. They are defined by specifying the border in terms of lines and circles Flat features may only be at Z = 0, and therefore all coordinates need X and Y only. In border segments, a radius of zero indicates a straight line, a positive radius means a circular arc that curves in a clockwise direction, and a negative radius means a circular arc that curves in a counter-clockwise direction.

```
struct horizontal_flat_feature
    }
    int           feature_type,
    int           ground_type,
    int           number_of_border_sections,
    xy_coord      starting_position,
    border_seg    border[n],
    {.

struct border_seg
    }
    int           radius,
    int           ending_x_coordinate,
    int           ending_y_coordinate,
    {.
```

<u>Horizontal</u> <u>Segmented</u> <u>Features</u>

These are road and stream features. These features are made up of segments, each of which is a trapezoid. If there are n segments, then there are n+1 sets of coordinates. The first set is the starting positions, one for each side of the feature. The remaining sets are two positions for the endpoints of each additional segment. Horizontal segmented features may only be at Z = 0.

```
struct horizontal_segmented_feature
    }
    int           feature_type,
    int           ground_type,
    int           number_of_segments,
    xy_coord      positions[2][n+1],
    {;
```

Vertical Segmented Features

This is the fence feature. A fence is made up of segments, each of which is rectangular. A fence has a starting position and then one position for the endpoint of each additional segment. All segments have the same height. These features are opaque and have no thickness.

```
struct vertical_segmented_feature
    {
    int         feature_type;
    int         color;
    int         height;
    int         number_of_segments;
    xyz_coord   positions[n+1];
    };
```

Compound Segmented Features

These are the rivers and trench features. Each segment of a compound segmented feature consists of a horizontal segment that represents the river or trench bottom, and two segments that represent the riverbank or trench walls. For each segment end, six positions are required, two for the corners of the bottom, one for each wall, and one for the water level on each wall.

```
struct compound_segmented_feature
    {
    int         feature_type;
    int         wall_ground_type;
    int         bottom_ground_type;
    int         bottom_depth;
    int         water_depth;
    int         number_of_segments;
    xy_coord    positions[6][n+1];
    };
```

If a river/stream is fordable, it must be a compound segmented feature, in order for the simulation to determine if the bank is navigable, otherwise, it can be a horizontal segment feature, which is simpler to render

Hills

Hills are represented as simple spherical sections, and so are essentially defined in terms of a radius and the position of the center. Variations across a single contour are always oriented as horizontal bands ("tree lines").

```
struct hill
    {
    int         feature_type;
    int         radius;
    int         width;
    int         height;
    int         treeline_height;
    xyz_coord   position;
    };
```

Spherical Patches

Spherical patch features allow parts of hills to be given specific properties. The feature is defined by giving the positions of the vertices of a bounding polygon. All points must lie on one hill. The lines between vertices will be great circles.

```
struct spherical_patch_feature
    {
    int         feature_type;
    int         ground_type;
    int         number_of_segments;
    xyz_coord   positions[n+1];
    };
```

5.1.4.3 Active Elements

Active elements are kept separately from the terrain, since the information about them is updated much more frequently.

Tanks and Other Vehicles

Tanks and other vehicles are cases of the simple vertical features. The graphics system knows how to display a tank when given the position and orientation information. The position and orientation are contained in the vehicle state table, which is updated by the network interface from vehicle position messages. An image of a tank consists of the tank body, turret, and gun barrel.

Muzzle Flashes

Tank gun muzzle flashes are visible from any line-of-sight position within 135 degrees of the gun centerline. These flashes are transient features, with a 1-second lifetime that begins at the instant the gun is fired. The position, depiction, and timing of muzzle flashes are available to all game stations from the firing vehicle's "shot" message.

Tracers

Tracers are visible only from the firing vehicle itself, they are transient features that follow the trajectory of the ordnance being fired. Computation of tracers' position, depiction, and timing is carried out by the firing vehicle.

Explosions

Explosions are visible only from the firing vehicle itself, they are transient features, with a short lifetime that begins at the instant the ordnance strikes a terrain element. Computation of an explosion's position, depiction, and timing is carried out by the firing vehicle.

5.1.4.4 Other Graphic Features

Texture Points

As explained above, texture points are locally generated spots that act as depth cues. Positions of the 1000 points will be kept in 10 by 10 by 10 array. Two of the dimensions represent X and Y positions in 20m increments, while the third dimension indexes the 10 points in each square. Every time the vehicle moves 20m in X or Y, a row or column of the array is updated with new positions for each point. For display, the graphics processor is passed a pointer to the array, which it considers to be a list of 1000 texture point features.

```
struct texture_points
    {
    int         feature_type;
    xyz_coord   position;
    };
```

5.1.5 Terrain Data Storage Estimates

Estimate of the number of objects in a 1km by 1km square.

Object	Bytes/object	Num objects	Total bytes
Bushes	10	200	2000
Trees	10	200	2000
Rocks	10	100	1000
Buildings	10	10	100
Hills	16	5	80
Road (10 Segments)	10+4(n+1)	2	108
Stream (10 Seg)	10+4(n+1)	1	54
River (10 Segments)	12+12(n+1)	1	144
Pond (10 Segments)	10+6(n)	2	140
		521	5626

The local terrain patch is 100 1km by 1km squares, so it would contain 52,100 objects and would need 562.600 bytes of storage.

A terrain patch update consists of a row of 10 1km squares. This works out to 5210 objects taking up 56,260 bytes of storage.

The full terrain database is 10,000 1km by 1km squares, so it would contain 5,210,000 objects and would require 56,260,000 bytes of storage.

Since there are 10,000 1km squares in the full terrain database, the size of the terrain database is very sensitive to the number of features in each 1km. The following table shows the effect on the full database of adding one object of each type to each 1km square.

Object	Bytes/object	Total bytes
Bushes	10	100.000
Trees	10	100.000
Rocks	10	100.000
Buildings	10	100.000
Hills	16	160 000
Road (10 Segments)	10+4(n+1)	540.000
Stream (10 Seg)	10+4(n+1)	540.000
River (10 Segments)	12+12(n+1)	1,440.000
Pond (10 Segments)	10+6(n)	700.000

5.2 Graphics Software Design

5.2.1 Overview

The game station graphics software operates on terrain and vehicle data to produce 3-D perspective color displays for the game station crew in real time The software resides in, and controls. the various processors making up the game station graphics hardware subsystem.

5.2.2 External Interfaces

Data available to the station graphics software is of four types: passive data, active data, station data, and internal data. Together these data form the "graphics data base." Passive data and active data are supplied over the network. station data is written directly into the graphics data base by the simulation software. internal data is written directly into the graphics data base by the graphics software itself. The contents of the graphics data base are controlled by the Graphics Data Base Manager process, described below.

5.2.2.1 Passive Data

Passive terrain data is supplied by the Terrain Server to the Graphics Data Base Manager, which updates the graphics data base. The data is organized in 1km by 1km squares, and is stored in a two-tiered tree structure consisting of (1) pointers to squares, and (2) the terrain data making up the squares. Terrain data is made up of template parameters, as described in section 5.1.4.

5.2.2.2 Active Data

Active data is supplied by the vehicles themselves directly over the network to the Graphics Data Base Manager, which writes it into the graphics data base. Once acquired, active data has the same form as passive data but is stored in a separate list. Vehicles provide data only when their conditions change, so the graphics software must update their frame-by-frame depiction; this is done by the Active Data Element Manager, described below.

5.2.2.3 Station Data

Station status is represented as a data block within the graphics data base. Some of the contents of this data block is generated by the simulation, and the rest by the graphics software; the entire data block is accessible to both the simulation and the graphics software. The lists below show the contents of the station data block; the numbers in parentheses indicate the order in which the data is filled in.

The data provided by the simulation process includes:

1. X and Y components of position.
2. Rotation angle (compass heading) of vehicle body.
3. Rotation angle (compass heading) of vehicle turret.
4. Elevation angle of gun barrel.
5. X and Y components of vehicle velocity.
6. Signal when a round is fired, including:

7. Type of ordnance
8. Table of (X,Y,Z) positions of round versus elapsed times since firing.

The data provided by the graphics software includes.

9. Z component of position.
10. Z component of vehicle velocity.
11. Terrain surface type
    If a round is in flight
12. Table of possible targets in the line of fire, terminated by the first such target that is a passive terrain element.
13. Which possible target is actually hit.

5.2.2.4 Internal Data

Internal data is generated locally by the graphics software and written into the graphics data base, thereafter it is treated as active data. Examples of internal data are: texture points, muzzle flashes, tracers, explosions, and dust clouds.

5.2.3 Flow of Control

The sequence of operations performed by the graphics software is summarized below. This sequence is performed every 200 msec, except as noted.

1. Read the portion of the station data block provided by the simulation. Complete the station data block. Compute the unity window and gunsight viewpoints. Delete all display lists.

2. Update the active and local data elements (vehicles, explosions, etc.).

3. Scan the entire graphics data base (every 1-2 seconds).

a) Flag each object large enough to be visible through a unity window at the vehicle's present position.

b) Flag each object within the gunsight viewport.

4. Scan the entire graphics data base.

a) For each object flagged in step (3a), perform viewpoint matching to each unity window. If the object could be visible in a window, put it into that window's display list.

b) For each object flagged in step (3b), perform viewpoint matching to the gunsight. If the object could be visible in the gunsight, put it into the gunsight display list.

5. Perform scaling and translation on all display lists. each object now has a size, location, and orientation relative to its view.

6. For each display list, expand objects into icons or polygon groups, depending on size. Prior to this step, each object has been treated as a single point.

7. Scale and clip all key points in the polygon groups added in step (6).

8. Sort each display list by range, for drawing according to the "painter's algorithm."

9. Scan-convert ("rasterize") each display list into image memory for display.

5.2.4 Software Organization

5.2.4.1 Overview

The above sequence of operations is performed by software resident in the various processors that make up the station graphics hardware subsystem. This section describes the individual software components and their functions.

All software will be implemented in the C programming language and in the microcode language appropriate to each processor. The Adage processors (graphics generator, MA1024) will be programmed using a commercially available cross-compiler software development package. The graphics MC68000 will be programmed using the existing BBN development package, consisting of a MC68000 C language compiler, linker, and debugger.

5.2.4.2 High-Level Graphics Software

The graphics MC68000 performs steps (1) through (4), processing the graphics data base to create new display lists every 200 msec. The graphics MC68000 shares some memory with the simulation MC68000, the station data block portion of the graphics data base resides in this memory.

The output from the graphics MC68000 is a set of sight display lists that is sent over the Adage 3000 bus to the graphics processor.

The graphics MC68000 software components are described below.

<u>Network Interface Process</u>

The network interface process handles asynchronous network traffic between the terrain server and the station graphics subsystem. It originates requests for terrain patch updates, and receives the updates when they arrive.

Graphics Data Base Manager

All access to the graphics data base, except for access to the station data block, is handled by this process, in the interest of speed. the station data block appears as an area of memory shared by the simulation software and the graphics software. The graphics data base manager performs storage and retrieval functions, applies terrain patch updates, and manages memory space.

Station Data Block Updater

This process provides information to the station data block by integrating the simulation-supplied data with terrain data. Using the supplied X and Y vehicle position components, it determines the underlying terrain feature and returns the corresponding Z position component and the surface type. Similarly, it returns the Z component of vehicle orientation (slope). Processing the supplied state vector, it returns a list of objects in the path of the projectile at the current time (the simulation determines which is actually hit).

Active Data Element Manager

Active data is supplied by vehicles and other active elements over the network, notifying the station of changes in parameter values. The active data element manager keeps their instantaneous position and orientation up to date by integrating the supplied velocity information.

Local Data Element Generators

These software components generate graphics objects parametrically, with no external input, and add them to the graphics data base.

Texture Point Generator

Texture points appear at random on the ground within one hundred meters of the vehicle, and provide the depth and motion cues corresponding to detail surface irregularities in the real world. This process controls the placement and positioning of the texture points.

Gun Effect Generator

A muzzle flash is visible from all vehicles within 270 degrees of the gun axis when a vehicle's gun is fired, this software process generates a muzzle flash, appropriate to the type of round in use, for every vehicle requiring it.

Tracer and Explosion Generator

Tracer rounds, and the visual effects of projectile strikes, are visible only from the firing vehicle itself, this software process generates these effects as needed.

First Scan Process

This process performs step (3) described above.

Second Scan Process

This process performs step (4) described above.

5.2.4.3 Low-Level Graphics Software

The graphics processor controls the translation of the set of display lists into displayable images, i.e. steps (4) through (9). It actually performs steps (6) and (8) itself. The instructions necessary to expand the template representations of data elements reside in the graphics processor's local memory.

The graphics processor software components are described below.

Task Manager

The task manager controls the order of execution of the tasks performed by the graphics processor. its primary function is to mediate the data flow between the graphics processor and the MA1024.

Object Expander

The object expander uses template algorithms built into its structure to expand the object parameters in the display lists into complete object representations, bitmaps or polygons, depending on object type and range.

Range Sorter

This process sorts the expanded display lists into range order for drawing according to the "painter's algorithm." The range sorter is invoked after all objects are expanded and matrix transformed, to prevent visual precedence artifacts.

Scan Conversion

Each display list passes through the scan conversion process, which converts the high-level graphics directives such as "draw a polygon" into bit patterns in image memory.

MA1024 Software

The MA1024 processor rapidly performs the matrix manipulations necessary to perform translation, rotation, and scaling, steps (5) and (7). Its resident software operates on display list data provided by the graphics processor, and is organized as a single microcoded process.

5.3 Graphics Hardware Architecture

5.3.1 Graphics Requirements

The graphics portion of the M1 simulation is the most computationally intensive part of the simulation. The graphics processing is described in more detail in Section 5.6 above, but it is summarized here to help explain the requirements for the graphics hardware. Briefly recapping, the scheme that will be used to generate the displays involves:

1. Visibility scan. Examining approximately 50,000 terrain objects to determine which are large enough to be visible through one of the unity windows. This operation happens

periodically (once every second or two) and produces a list of about 5,000 visible objects.

2. Bucket sort. This smaller list is then examined, and any object that may appear in one of the seven unity windows is put on that window's display list. This happens once every 200ms.

3. Transformation. These seven lists are then transformed to the sight viewpoints.

4. Icon distinction. The objects are sorted into icons or are expanded to define a complete polygon centered about a single point.

5. Scaling. The completed polygonal objects are scaled for perspective.

6. Range sort. Each window display list is sorted by range for drawing with the painter's algorithm.

7. Rasterization. Finally, the display list is rasterized into the image memory and displayed.

A similar process takes place for the gunner's sight, but all objects within the 20-degree (6 degrees at 10x) field of view must be examined becausee of the increased range of the sight using either 3x or 10x magnification.

It was originally believed that it would be possible to use low resolution displays (256 x 256) for scene presentation. However, with this low resolution, a 10-meter object at 1km occupies only 1 pixel when one display provides a 60-degree field of view. To provide a very high quality image, a 4096 x 4096 resolution display would be needed. The cost of developing such a system is well beyond the scope of this project. For this reason, we propose that a 512 x 512 resolution display be used, and that important objects (potential targets, like tanks) be anti-aliased to provide better effective resolution.

The seven steps outlined above must be performed five times a second to provide a sufficient level of animation for this simulation. This allows only 200 milliseconds to produce all eight displays. Each display must have a double-buffered image memory so that a new image can be generated without affecting the image being displayed.

Current estimates show that graphics database management (tasks 1 and 2 above) will require 67% to 75% of a 10MHz 68000 processor. Transformation and perspective division (tasks 3 and 5) require many multiply operations. A 68000 is not well suited to this task, nor are any other available microprocessors. Thus, it will be necessary to have special-purpose hardware to perform these operations. A 68000 could perform a range sort (task 6), but not in the time available. A high-speed microcoded processor would be applicable here. The rasterization for eight displays (task 7) is well beyond the capabilities of a 68000 or any other generally available microprocessor and will require specialized graphics hardware. A pixel painting speed of 20 million pixels per second is necessary to paint eight displays that have a resolution of 512 x 512 five times a second, using a painter's algorithm. A high-speed microcoded processor would be particularly effective for this application.

5.3.2 Possible Solutions

Given the various technical requirements of the project, schedule, and budget constraints, we believe that the Adage 3000 offers the best building block for the project. The Adage 3000 is well proven, with over 100 systems in the field. The basic Adage offering will allow software development tasks to begin while special enhancements to the Adage hardware are designed and implemented. This is not necessarily the least expensive option, but with such limited development time, we feel it is the only practical alternative. For a discussion of hardware alternatives, see [Gurwitz, 1983].

The Adage 3000 consists of a high-speed synchronous backplane with a wide selection of modules available to produce a graphics system of the desired characteristics. For the SimNet project, a subset of the available modules will be used. Figure 2 is a block diagram of a typical, standard Adage system. In Figure 3, a block diagram shows the Adage system that will be used for early software development. The GPS module, which is a

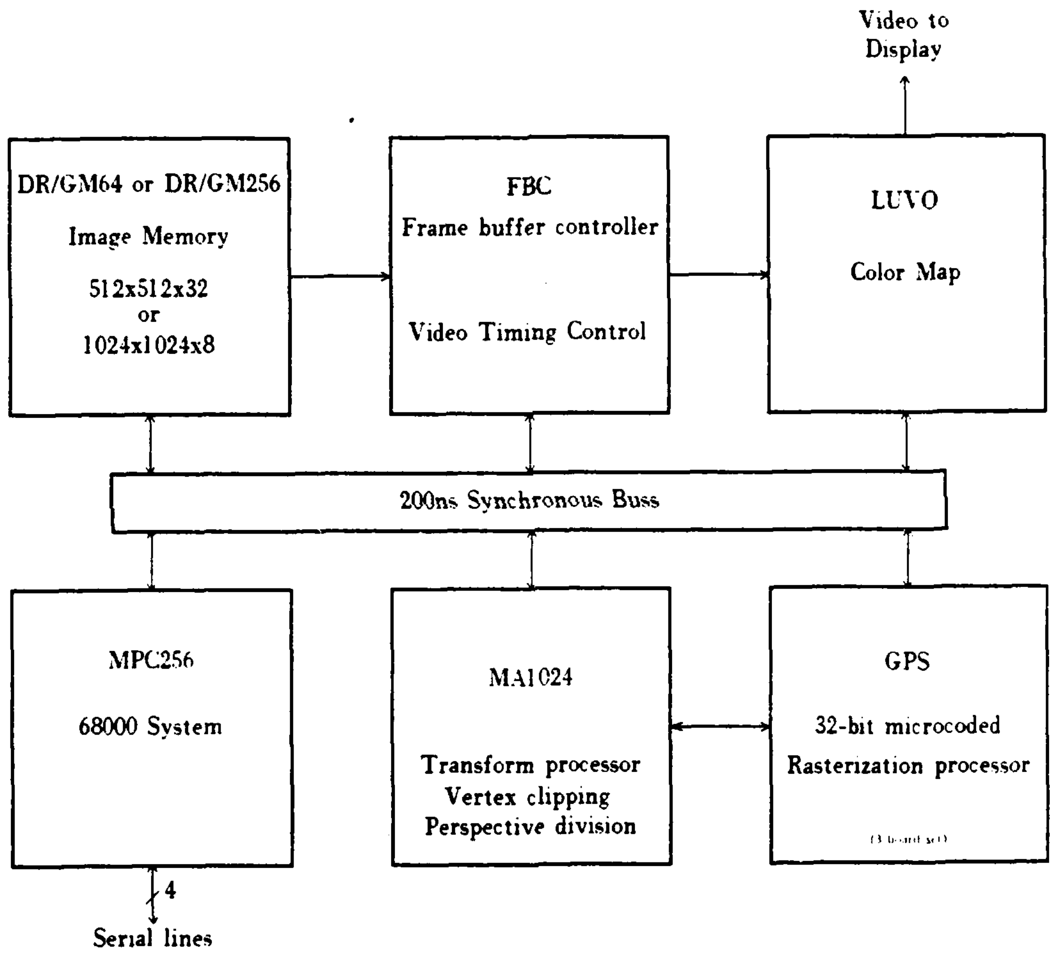


Figure 2. Standard Adage Architecture

microprogrammed 32-bit machine, will be used to perform icon distinction, range sorting, and rasterization (tasks 4, 6 and 7). The GPS will be used only until the AGG4 is available (November 1983). The AGG4 is also a microprogrammed processor, but is one-third the cost of the GPS and is better suited to the rasterization task (7). The MA1024 is a microprogrammable transform processor. It will be used to perform tasks 3 and 5. The GM64/256 modules provide rasterized image memory. The FBC in unison with the LUVO produce the video stream to a 1024 resolution display.

However, the Adage 3000 itself does not fulfill SimNet graphics requirements. The Adage system typically drives only one display (512 x 512 or 1024 x 1024 resolution), while SimNet requires eight independent displays. It also has other shortcomings, such as a lack of general-purpose processing capability and network interfaces. Solutions to these deficiences will be discussed below.

5.3.2.1 Image Memory and Video

The GM256, FBC, and LUVO are designed to drive a single display, while SimNet requires eight independent displays. The straightforward solution is to replace the GM256/FBC/LUVO combination with four copies of a newly designed board. Each board contains the double buffered image memory for two 512 x 512 x 8 displays, as well as the color lookup tables and video DACs. (The color lookup tables and three DACs are available as a single 2" x 2" hybrid package.) This produces a very modular, clean solution. It results in a board potentially much less expensive than the Adage equipment, which can be installed in a quantity sufficient for the number of displays required by the simulator.

To manage all of the graphics tasks, a second AGG4 must be added to the configuration. One AGG4 will be dedicated to the rasterization task (7), while the other will handle tasks icon distinction and range sorting (4 and 6). See Figures 4 and 5 for block diagrams of this system configuration and this new board.

This solution involves designing a memory board that can be accessed in the same manner as the Adage GM256. The interface on the memory board must allow a color to be set and pixels to be written as specified by a 32-bit-wide mask. All bits of each pixel are written as a result of this single command.

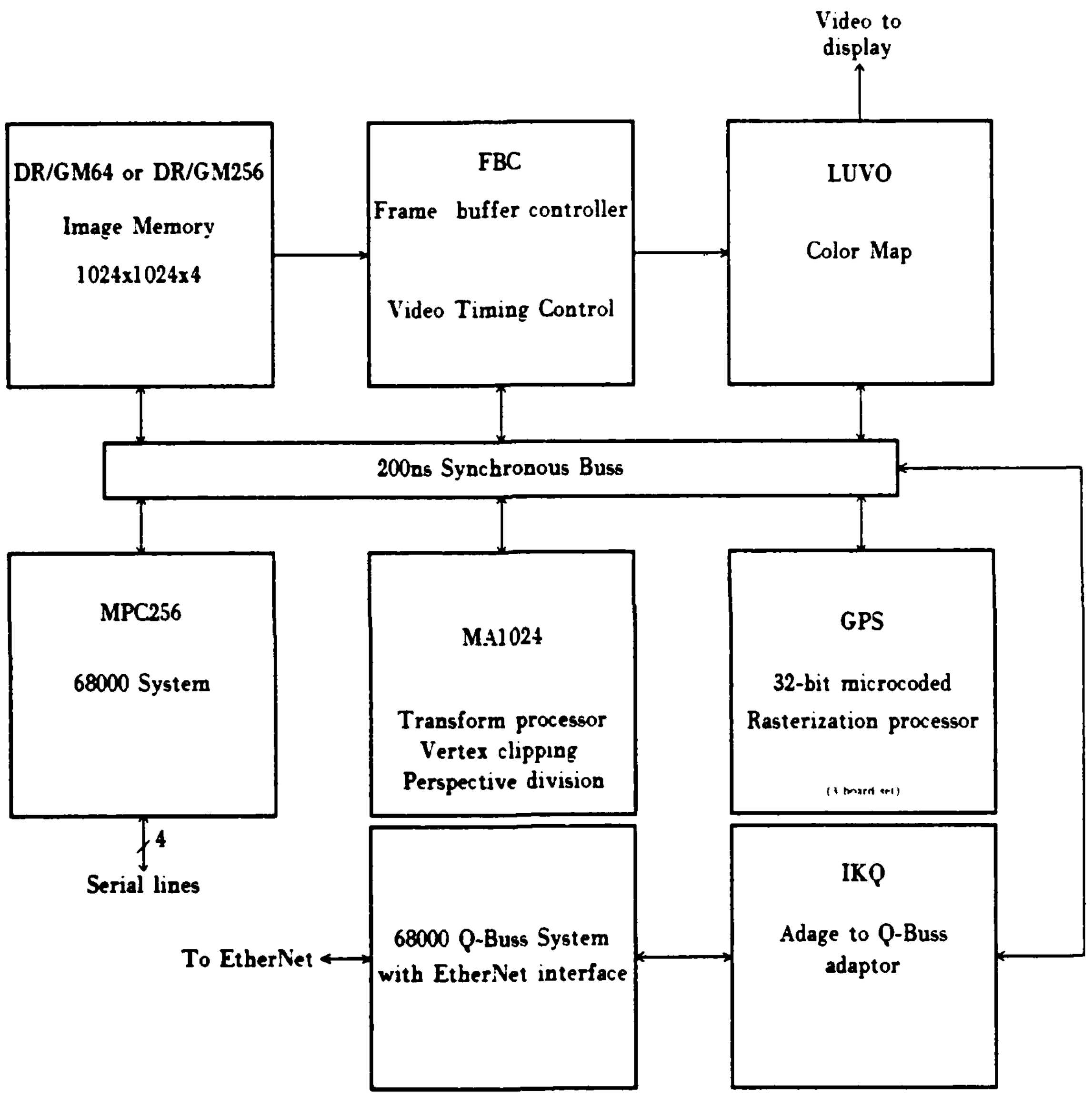


Figure 3. Development System

This is the preferred solution given the cost/time constraints of the SimNet project. It offers a good compromise between development time, cost per copy, and required performance.

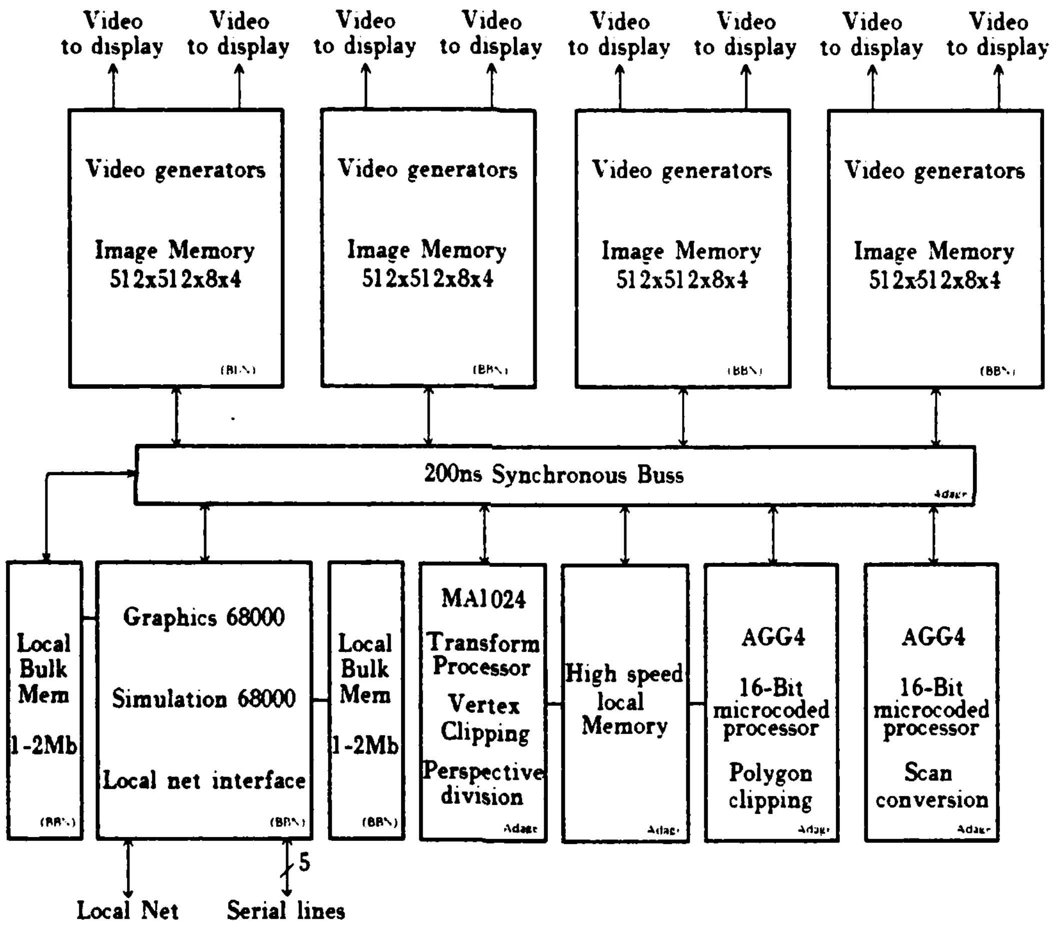


Figure 4. GS Block Diagram

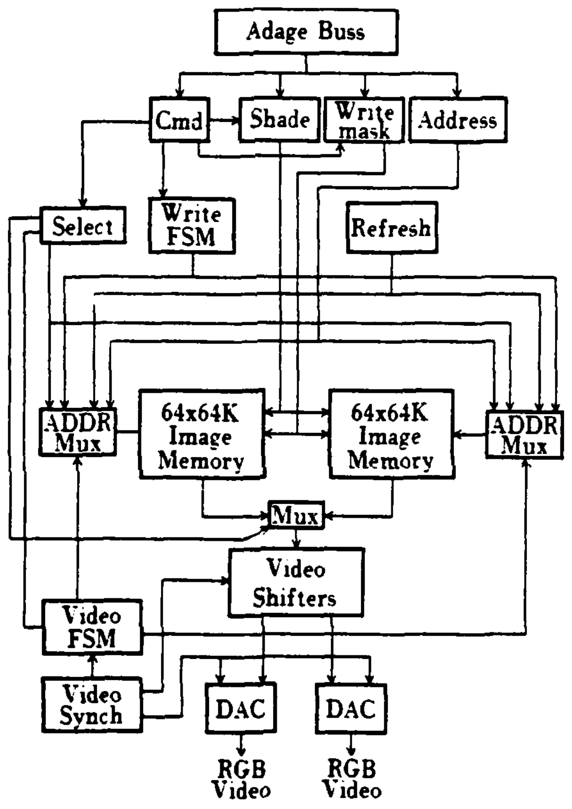


Figure 5. Image Memory and Video Block Diagram

5.3.2.2 Intelligent Video Controller

To reduce cost and increase flexibility, an alternative solution can be proposed. This solution has higher development costs, but yields a result better able to deal with the rasterization process and possible later expansion of the system

The Adage 3000 is again used as the backbone of the system, but no GM256/FBC/LUVOs, and only one AGG4 are required. The difference from the previous solution is that instead of designing a simple video controller with image memory, a more ambitious display subsystem is envisioned.

This subsystem would have double buffered image memory for one 512 x 512 x 8 display, color lookup table, and video DAC. In addition, this display controller would have a high-speed microcoded processor on board. Each processor would be responsible for rasterizing its own display image. This scheme spreads the 20 million pixels per second rasterization task over eight processors. The AGG4 need only pass high-level drawing commands to each display controller. See Figure 6 for a block diagram of this board.

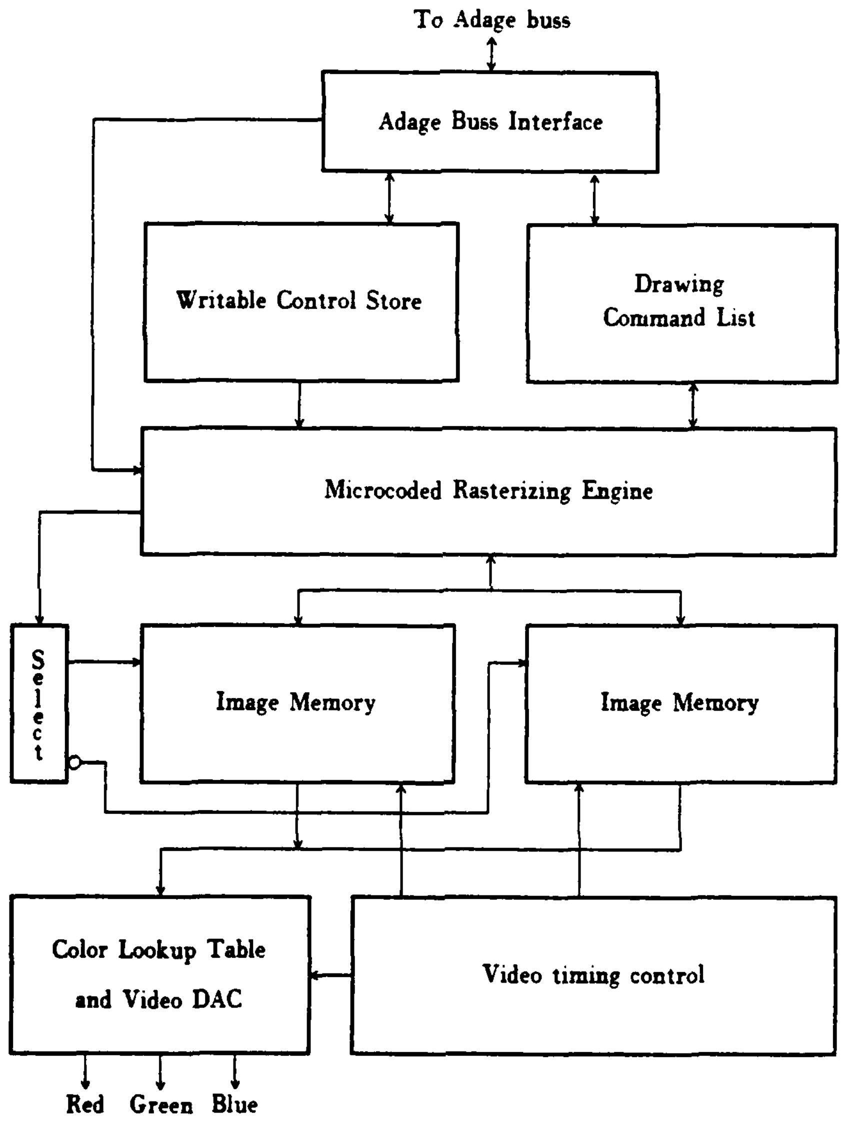
This solution will fulfill the needs of the currently planned system in terms of the complexity level of the displays. It will also have sufficient cycles left over to permit us to consider more ambitious display schemes once the initial system is in place. The intelligent video controller is a substantially more ambitious effort than the previously mentioned solution. This means a longer development time and greater development risks.

A modular image generation scheme will allow the hardware for follow-on simulators (M2/M3) to be tailored more effectively to the needs of each. Having the ability to generate eight displays when only four are needed could cost a substantial amount of money.

5.3.2.3 Video Splitter

The solution with least development cost and risk was arrived at during consultations with Adage. In these discussions, Adage felt that standard Adage 3000 products could provide 90% of the graphics solution.

Four GM256 image memory boards can be used to provide sufficient memory to double buffer eight 512 x 512 x 8 displays. Two AGG4 processors in conjunction with an MA1024 multiplier could be used to perform the perspective transformations.



Intelligent Video Controller
Figure 6

clipping, and rasterization functions.

A 10MHz 68000 (from Adage, or available on a Q-Bus) would then be used to perform the graphics database management. Calculations show that such a configuration would be capable of maintaining eight displays at a five-frame-per-second update rate.

This solution requires the design and implementation of a new video controller board. It would pull the raster image from the GM256s, pass the bits through color lookup tables, and produce eight RGB video outputs. This is not a complex board to design or implement.

The disadvantages to this solution are high cost and low flexibility. Without commitments to quantity purchases from Adage, the cost per M1 simulator will remain relatively high. Even with quantity discounts, the per-system cost may be higher than desired. In addition, this solution is not particularly flexible. It will probably not be able to drive more than eight displays, and if it is driving fewer displays (as may be true in the M2/3 simulators), it will require virtually the same hardware with little savings in cost. Also, little "headroom" is available for increased display complexity if desired.

5.3.2.4 Conclusions

The image memory with video generators is the most practical solution for the present system. The intelligent video controller would be a good extension for follow-on systems, but as a practical matter, there is not sufficient time to develop this subsystem for the current project. The video splitter offers little flexibility and costs too much to be considered a practical alternative.

5.3.3 Hardware Design

5.3.3.1 Image Memory and Video

Each Adage-compatible board will contain hardware to generate two independent 512 x 512 x 8 displays. There are two reasons for putting two display generators on each board. The first is to reduce the number of boards in the Adage backplane. The second is to improve effective memory cycle times. This can be done by interleaving the two displays within each of the two image buffers. Only one display will be written at a time since the rasterizing AGG4 will be dedicated to processing a single display until it is complete. This is an artifact of the bucket sorting and eliminates any contention in writing the image memory. On the video side, the other image buffer will be read to produce the video stream. The two displays will be running with the same sync generator and will be running in tandem.

The memory required to represent one display (512 x 512 x 8 bits) can be held in 32 64K DRAMs. The AGG4 is capable of writing 32 pixels at a time. Each of these pixels is eight bits in depth. Obviously, it is not possible to write 32 x 8 bits into 32 chips in one write cycle. However, by using the page mode access features of DRAMs and cycling the memory chips eight times, the write requested by the AGG4 can be performed and will take 1.6 microsecond. The AGG4 can produce write requests at a rate of about 1 per microsecond while filling a polygon. Clearly, the memory will slow the down the AGG4. The problem can be solved by cycling 64 DRAMs on each write. Since each display is double buffered, 64 chips are available. In this configuration, each write from the AGG4 will cause four cycles of the image memory (800ns). Each cycle will write two bits of each of 32 pixels. These writes will be done in page mode by hardware on the memory board. However, this also means that memory cycles must be shared with the video generator, thereby producing contention for the memory. Without the video requests for memory cycles, this configuration could respond to AGG4 requests once every 800ns. But the video section requires a memory cycle every 3.2 microsecond, or every fourth memory cycle. Therefore, the average memory access time is 1 microsecond. This matches the average AGG4 write request time.

An average memory cycle time of 800ns can be obtained if the video interference can be eliminated. This can be accomplished if two display generators exist on one board. With the additional 64 DRAMs, the memory can be configured to avoid contention with the video generator. One buffer can be dedicated to the video generators while the other serves the AGG4, thereby eliminating interference. Each of these buffers comprises two displays.

From the video side, for a 512 x 512 display, the pixel time is roughly 100ns. The memory will be able to produce 32 pixels every 750ns. This is four times the bandwidth required for one display and will easily accommodate two displays. The video section will cycle the memory eight times in page mode to supply 32 pixels for each of the two display streams.

It will take four person-months to design this subsystem. Board fabrication will require another six weeks of elapsed time, and checkout of the prototype will take one person-month.

5.3.3.2 Intelligent Video Controller

The intelligent video controller improves the system's rasterization capabilities by almost an order of magnitude. This is accomplished by using eight rasterizing processors rather than one.

Each video channel would have its own rasterizing processor, double-buffered image memory, color lookup table, and video DAC. To accomplish this, a microprogrammed processor (AMD29116) with 4K words of microstore is required. Another 4 to 16K words of command buffer is also required to accept commands from the AGG4. An Adage bus interface, two image buffers, video timing, color lookup table, and DAC are also required. Figure 6 is a block diagram of this board.

This microprogrammed system would have a basic cycle time of 100 to 125 ns. The 29116 is well suited to tasks like rasterization. It has an integral barrel shifter and the ability to merge data with the output of the shifter. These are exactly the types of capabilities required for rasterizing images. To further improve performance, the memories would allow write operations of the kind described in the previous solution. This would allow the 29116 to perform one write to memory that could modify as many as 16 pixels at a time. Each pixel is 8 bits deep, so the memory would perform multiple cycles to effect the single write command issued by the 29116. This will make more 29116 cycles available for considering polygonal edges, etc.

This effort is considerably more complicated than the previous solution. The design effort would require eight person-months: seven for hardware design, one for initial diagnostic microcode. An additional three months is required for contruction of a simulator for microcode development. Prototype board construction would take ten weeks and checkout would take two months.

5.3.3.3 Video Splitter

As described above, the video splitter is possibly the simplest of the three solutions to the graphics problem. The most difficult part of this design is understanding the mappings between the Adage GM256 memories and the actual video stream.

This design will take roughly two person-months to determine the correct mappings of memory to video and understand the existing Adage buses. Another 1-1/2 months will be required for the actual design, six weeks for board fabrication, and 1 month for checkout.

6 Networking

There are two major areas of interest in the networking portion of the SimNet design. the local area network (LAN) and the long haul network (LHN). In the first phase of SimNet implementation. the LAN is all that is required. It ties together nearby stations with high data throughput and low delay. It is used both for passing state vector and projectile information between stations, and for sharing the terrain database resources. The requirements and design of the LAN are discussed in detail.

The LHN is not required in Phase I, but is crucial in later phases to the goal of having large numbers of participants that are widely separated geographically. Since the LHN is not implemented until later phases of the SimNet program, a detailed design is not presented here. We do present a detailed design study including requirements for the LHN imposed by the LAN design, and possible alternatives for implementing and configuring the LHN to meet these requirements. As part of the LHN aspects of the design, a Communications Interface (CI) processor is required to connect the LAN to the LHN. Again, a detailed design of the CI is not presented. Rather, we present a functional description of the CI and technical requirements for its design.

6.1 Network Requirements

The network for SimNet must support two operating modes. First, it must be able to broadcast a large number of small packets at periodic intervals (5 times per second). and second, it must be able to make asynchronus point-to-point transfers of relatively large packets. In addition, these two modes must be in operation concurrently. The following sections describe the types of network traffic (local area and long haul) that must be supported, and the traffic characteristics of each of the processor types on the local area networks (LAN)

6.1.1 Vehicle State Vector Update

Five times a second, each vehicle simulation provides an updated state vector to the network. In the same interval, the simulation must receive all state vectors from vehicles whose coordinates put them within the local terrain patch of the local simulation.

The long haul Communication Interface (CI) also needs to receive all the state vectors generated locally. It then sends them immediately to all the other CIs. Whenever the local CI receives state vectors from any other CI, it puts all of them on the LAN at the next 1/5 second interval.

6.1.2 Terrain Map Update

The terrain map update will be sent only by the terrain map server, and will be received in a given instance only by a particular vehicle simulation. The update calculation is performed in the Game Station (GS), and an update request message is sent to the terrain map server, which will then respond with the terrain map update.

6.1.3 Vehicle Event Notification

Some events occur within a particular vehicle simulation that must be propagated to another vehicle (or all vehicles). An example of this is "hit" notification. After the vehicle has fired and performed the trajectory calculations, it determines if any other vehicle was hit. If so, it sends a "hit" notification directly to that vehicle. As in the case of the terrain map update, this is another example of point-to-point data transfer. The hit notification will go to either a GS on the LAN or to the CI, which will transfer it over the long haul network (LHN) to another LAN and on to the remote GS.

6.1.4 Global Time

It is important for all the vehicle simulations to be running on the same time base. Either some designated processor on the LAN (such as the Game Support Station) or the network hardware should provide the time base. Also, there should be a way of synchronizing the time base for the various LANs.

6.1.5 Session Initialization

The terrain patch for each vehicle and the initial state of each vehicle must be provided at game initialization time. At this time, the LAN is operated in the same mode as when it is doing terrain updates, that is, asynchronous point-to-point transfers.

6.2 Network Constraints

6.2.1 Local Area Network Bandwidth Analysis

Since all SimNet message traffic must be sent over the LAN, it has the most demanding throughput requirements. There are three types of messages that will use the LAN: terrain updates, simulation events, and vehicle state vectors. Of these three, the terrain updates will put the heaviest load on the LAN in terms of volume of traffic.

For ease of analysis, let us make some simplifying assumptions about the terrain patch updates to the simulation:

1. The vehicle is in the center of a 10km by 10km square terrain patch that is located completely within a 100km by 100km terrain.

2. Each time the vehicle moves foward by one meter, a 1m by 10km section of terrain appears in front and a section of identical size and shape disappears behind.

3. The number of bits required to describe a 1m by 10km section of terrain is 447. (This number was determined by dividing by 100 the number of bits required to describe a 1km by 1km area of the terrain. It is 10,000 square meters or two football fields in area.)

4. The theoretical maximum vehicle speed is 32m/s (72 mph) and the practical maximum is 20m/s (45 mph). (Typical vehicle speeds will be below the practical maximum for the simulation.)

As can be seen immediately from these assumptions, the throughput required to support the terrain updates for a single vehicle is a direct function of the linear speed of the vehicle. The bandwidth of the LAN must be high enough to support the terrain updates for all vehicles on the LAN at the maximum vehicle speed. (It is quite feasible at certain times during the simulation for all vehicles to be traveling at top speed, in a convoy, for instance.) The one variable left to be determined is the number of vehicles supported by the LAN. In this and in the following analyses, we will use 8, 64, and 256 as the numbers of vehicles in three LAN configurations. This will give some indication of problems of scale in the simulation network. Below are the bandwidth requirements for the three different LAN configurations.

Bandwidth = 447 * (20 m/s) * (local number of vehicles)

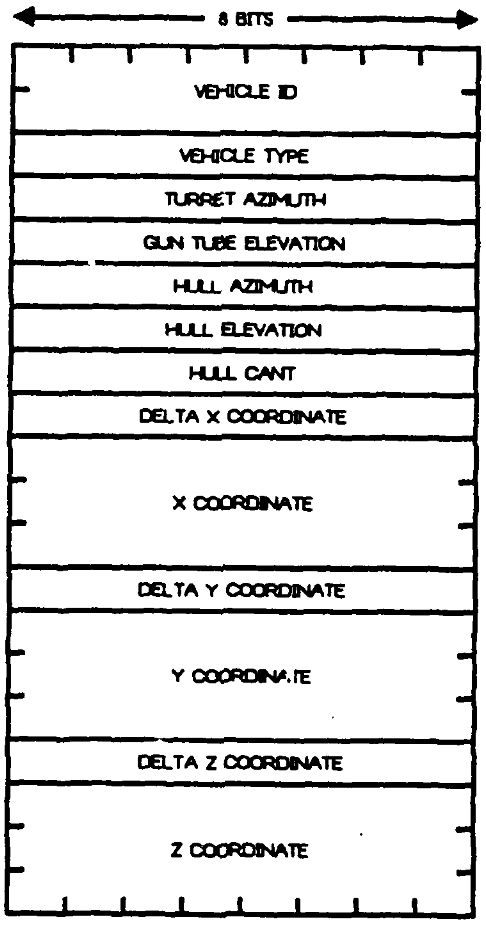
8 vehicles	.071 MBit/sec
64 vehicles	.572 MBit/sec
256 vehicles	2.288 MBit/sec

Although it appears that even the 256 vehicle case is well within current LAN technology (which provides 10 MBit/sec), we should emphasize that practical experience indicates that a sustained rate of throughput that is half the theoretical limit is often difficult to achieve. In light of this, we should be cautious about loading a single LAN with a large number of vehicles, bearing in mind that other types of message traffic must be supported as well. From this point on, we will assume that no LAN will be required to support more than 256 vehicles

The other types of message traffic that the LAN must support are the simulation event and state vector update messages. The simulation event messages are infrequent (relative to the 1/5 second update rate), and under normal and expected simulation conditions, will be an insignificant percentage of the traffic on the LAN. However, depending on two factors, the state vector update traffic could be significant on the LAN. These factors are the total number of vehicles supported by the simulation and the algorithm used to determine when state vector updates are sent. The discussion of these factors at this point is moot, because long before the state vector updates cause a problem for the LAN, they will have exceeded the capabilities of the LHN. Because of this, we postpone our discussion of the state vector update algorithms to the LHN sections.

6.2.2 Local Area Network Delay Analysis

On a 10 MBit/sec LAN, delay is not a problem with respect to the 200-millisecond time scale we are working with. To illustrate, 256 state vector update messages, each 160 bits long (see Figure 7), can be transmitted to every vehicle on on a LAN in less than 4 milliseconds. However, there are a couple of issues to be considered. First, there must be a guaranteed "update window" that recurs every 1/5 second so that the state vector updates can be transmitted without interference from other traffic. Second, it must be determined if it is necessary to artificially delay local area state vector update messages to match the delay characteristics of similar messages coming over the LHN.



Vehicle State Vector Update
Figure 7

6.2.3 Long Haul Network Bandwidth Analysis

As was indicated previously, the terrain patch update messages are exclusively a LAN phenomenon. The only messages traversing the LHN are simulation event messages and vehicle state vector update messages. The number of vehicle state vector updates per second is the driving factor in determining the necessary bandwidth for the LHN. This number is dependent on the algorithm used to determine when update messages should be sent by the simulation network interface. There are two basic alternative algorithms that trade off local computation within the vehicle and LHN bandwidth.

The algorithm that optimizes for local computation (position update algorithm) specifies that state vector updates contain the absolute vehicle location and that they be sent to every other vehicle every 1/5 second. Assuming a state vector size of 160 bits, we have the following bandwidth requirements.

Bandwidth = 5 * 160 * (total number of vehicles)

8 vehicles	6.4 Kbits/sec
64 vehicles	51.2 Kbits/sec
256 vehicles	204.8 Kbits/sec

It can be seen that the LHN bandwidth reqirements go up linearly as the total number of simulated vehicles increases. The bandwidth values given represent total throughput between LANs over a LHN. All state vector update messages must go to all other LANs, so the LHN throughput is dependent on the total number of vehicles.

The second algorithm (velocity update algorithm) requires absolute velocity in addition to location in the vehicle state vector update. If the vehicle is stationary or moving at a constant velocity, then state vectors need be sent at less frequent intervals (e.g., 3 times per second). At the receiving vehicle, however, for each 1/5 second the new location must be calculated from the last known position and velocity. This calculation must be performed for each vehicle that the local vehicle must display whose position information is obsolete. The bandwidths required by an update rate of 3 times per second are.

Bandwidth = 3 * 160 * (total number of vehicles)

8 vehicles	3.84 Kbits/sec
64 vehicles	30.72 Kbits/sec
256 vehicles	122.88 Kbits/sec

The velocity update algorithm gives us quite a bit more flexiblity with respect to LHN traffic. Since delta position information is provided in the state vector update, the simulation can use last known position and delta position to calculate the new position for the next 1/5 second interval, in the absence of a new state vector update. This means that a new state vector need not be sent every 1/5 second, but instead may be sent every 1/3 second. Referring to the previous bandwidth calculations, we note that it is possible to send more than 64 vehicles worth of state vector updates using the velocity update algorithm on a network using 56 KBit trunk lines.

The first algorithm will be implemented for earlier vehicle simulations when the number of vehicles is small, and all are on the LAN. Later, to relieve LHN congestion and delays, the second algorithm will be installed, if experiments prove it to be a viable alternative.

6.2.4 Long Haul Network Delay Analysis

Delay is longer in LHNs than in LAN by two orders of magnitude. Given that transcontinental distances are involved, we are talking about an absolute minimum propagation delay of 20 milliseconds. Add to this 30 milliseconds or more possible switching delay imposed by the common carrier, and the minimum delay is 50 milliseconds. If a 56 KBaud common carrier trunk line is used, then there is a 3 millisecond transmission delay for each 160-bit state vector update message that must be sent over that trunk line. The most favorable configuration for the LHN in terms of delay is a single trunk line (ignoring the possiblity of multi-trunk techniques for the moment) directly connecting every pair of geographically separated LANs (Figure 8). The following delay tabulation is for this configuration.

Delay = 50 ms + 3 ms * (local number of vehicles)

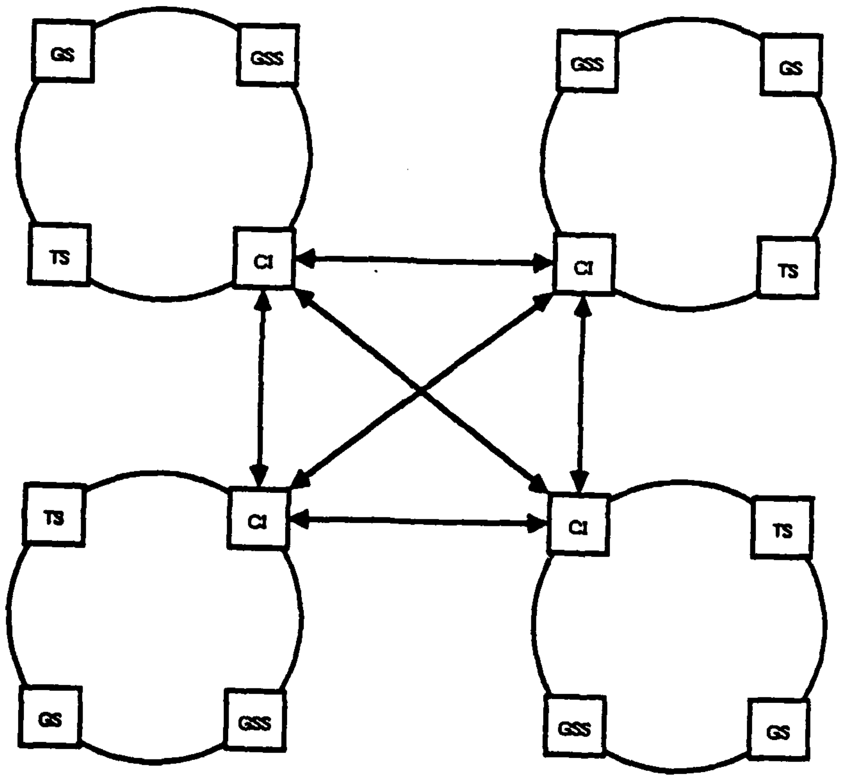
8 vehicles	74 ms
64 vehicles	242 ms
256 vehicles	818 ms

(The above table gives delays only on the LHN component. Delays between the vehicle simulation software and the LHN interface on the CI are not included.)

An alternative to 56 KBit/sec land lines is the satellite broadcast network. This technology introduces an entirely different set of delay characteristics. Using a 1 MBit/sec satellite channel we have a 250 millisecond propagation delay but only .16 millisecond transmission delay for a 160 bit packet.

Delay = 250 ms + .16 ms * (local number of vehicles)

8 vehicles	252 ms
64 vehicles	261 ms
256 vehicles	291 ms



LHN Configuration Example
Figure 8

6.2.5 Network Scalablity

We can define some limits to the scale of SimNet based on current network technologies. To do this, we must determine the upper bound for LANs and then discover the bounds for LHNs. First, let us examine the limits imposed by LAN technology. Making the assumption that we can use most of the bandwidth of a 10 MBit/sec LAN (see section on Local Area Network Implementation), what is the largest number of total vehicles that can be supported? If each LAN has 256 vehicles, then 2.5 MBit/sec must be subtracted to allow room for the terrain database and other miscellaneous traffic. We make the additional assumptions that we are using something similar to the velocity update algorithm so that updates are required at the rate of 3 per second. With a remaining usable bandwidth of 7.5 MBit/sec and a 160 bit state vector, we have an absolute maximum of 15,625 vehicles. This is the number of vehicles that will saturate a 10 MBit/sec LAN with state vector update messages.

Since there is no current LHN technology that will yield 10 MBit throughput, we are going to have to scale down our expectations. The long haul technology that offers the best throughput is the wideband satellite broadcast network, which will support 1 MBit/sec with reasonable reliability. At this bandwidth, the satellite channel will allow a total of approximately 2000 vehicles at 3 updates per second. We see, then, that network scaling for SimNet is currently limited by LHN technology.

Given that the SimNet network is limited by long haul bandwith constraints to 2000 vehicles, what is the realistic total number that can be supported? One consideration is that the actual throughput of the LAN is significantly less than the theoretical throughput. Since we are already limited to 2000 vehicles, we know that the state vector update traffic will only require 1 MBit/sec of throughput from each LAN. Even allowing that the actual LAN bandwidth is half the theoretical, it should still be able to handle the throughput requirements of the state vector traffic plus the terrain update traffic. We know then that the LAN will not be the limiting factor in this case.

There is high confidence that the satellite broadcast network can actually support 1 MBit/sec throughput. The network is designed to carry three types of service, unreliable at 3

MBit/sec, moderately reliable at 2 MBit/sec, and highly reliable at 1 MBit. Using the third type of service, we can expect to support on the order of 2000 vehicles for SimNet, at the cost of using satellite broadcast for the LHN.

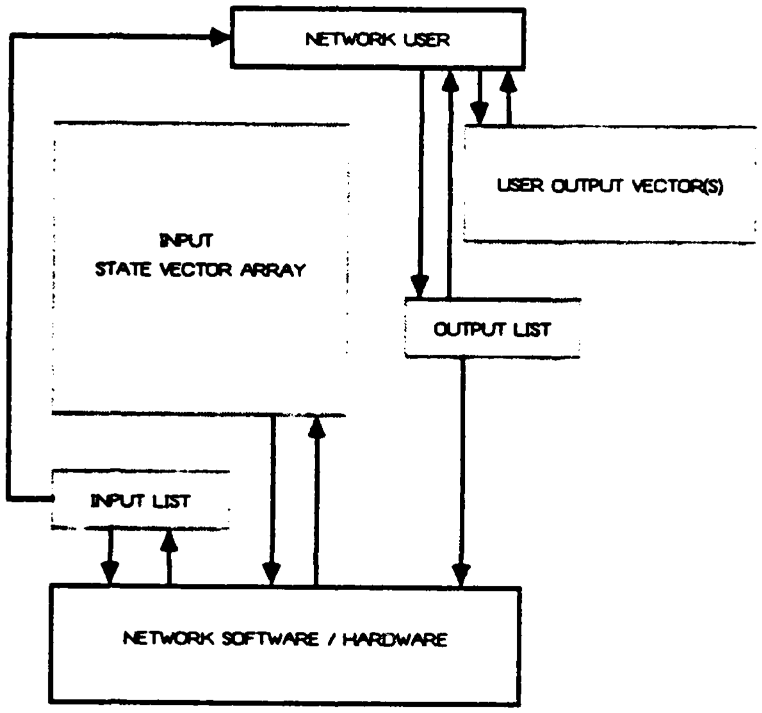
6.3 Network Interface

The network interface supports two modes of interaction, synchronous and asynchronous. The synchronous mode supports periodic transmission and reception of state vectors on the SimNet network. These state vectors are filtered at a particular SimNet processor for state vectors that are appropriate for that processor. For example, if the processor is a GS, the state vectors will be filtered for those whose coordinates fall within the terrain patch that the GS is working with. Alternatively, if the processor is a GSS, then only state vectors from the LAN will be accepted. All state vectors will be received by the network hardware, but only relevant state vectors will be passed to the network user.

The asynchronous mode supports general point-to-point and broadcast transmission of messages. These messages may be sent and received at any time by the network user.

6.3.1 Synchronous Operation

The synchronous mode requires seven parameters. an input list, an output list, an array of state vectors, size of state vectors, maximum number of state vectors, a filter selector, and the filter selector argument. These parameters are supplied with the SYNCH_INIT command. Other commands are SYNCH_START, SYNCH_STOP, SYNCH_GETSTATUS, and SYNCH_SETSTATUS. Once the SYNCH_INIT command is given, the SYNCH_START command may be issued, and the network will begin operating in synchronous mode (Figure 9).



Network Interface (Synchronous)
Figure 9

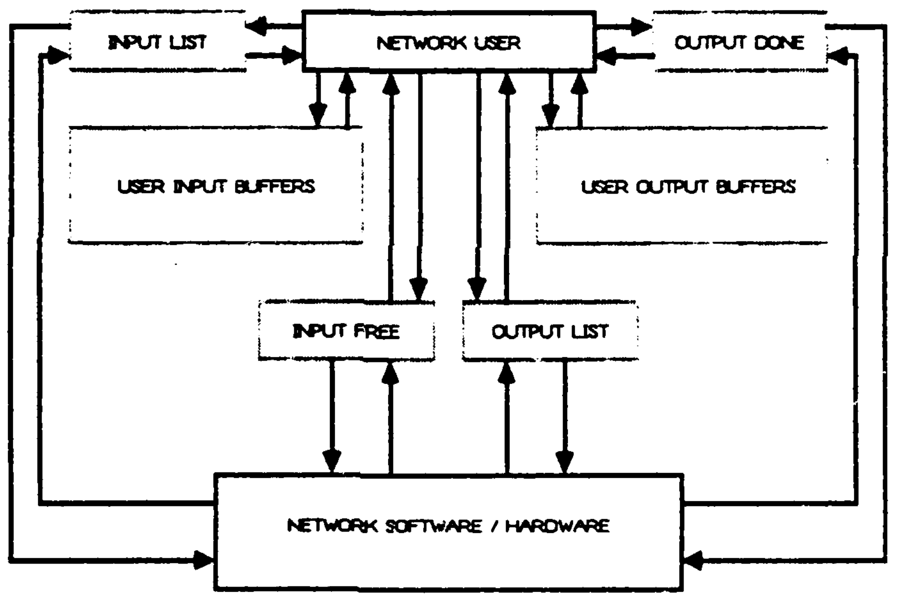
Once in operation, the network (software or hardware) will continuously "listen" for state vectors and for an indication to send the local state vector(s). When all of the state vectors are received into the state vector array (provided by SYNCH_INIT), a pass is made over all the vectors with the selected filter. All vectors that pass the filter are put on the input list if they are not already there, and all vectors that do not pass are removed from the input list if they are on it.

When the send indicator is received, the network sends all state vectors on the output list.

The network user receives notification when all of the state vectors have been updated, and when the local state vector(s) have been sent. This notification will be in the form of either a hardware interrupt or some type of software event.

6.3.2 Asynchronous Operation

In this mode, a number of buffers can be submitted for either input or output. The network user is notified when the operations are complete. The operations for this mode are ASYNCH_INIT, ASYNCH_RECEIVE, and ASYNCH_SEND. Arguments to ASYNCH_INIT are pointers to input and output done lists, an input free list, and an output list. The network signals the application software each time a buffer is placed on either of the 'done' lists. The application signals the network software/hardware each time a buffer is placed on an empty input free list or output list (Figure 10).



Network Interface (Asynchronous)
Figure 10

6.4 Network Implementation

6.4.1 Local Area Network Implementations

Since the first implementation that corresponds to the above discussion will be done with standard Ethernet hardware, most of the features described will have to be simulated in software. We believe that the Ethernet implementation will be adequate for any LAN configuration of less than 32 vehicles. However, for the final implementation of the LAN, we are currently considering a ring network.

The main reason for considering ring network technology over Ethernet is its delay characteristics [Bux, 1981]. As the load on the network increases, delay for an arbitrary packet increases linearly for a ring network and exponentially for an Ethernet. The Ethernet bases its contention scheme on collision detection, with exponential backoff for retransmission. Collision is most likely to occur when hosts are trying to transmit at regular intervals, as in the SimNet application. This can cause arbitrarily long retransmission delays for individual packets. Since delay under heavy loads is an importantant concern for this application, and the throughput for both types of network are roughly equivalent for the same range of load values, the ring network architecture is clearly superior.

6.4.2 Long Haul Network Implementations

At this stage of the project, a final design of the LHN would be premature. There are still unresolved issues, such as the algorithm to be used for updating the vehicle state. However, there are some options that can be investigated.

The first two phases of the SimNet implementation (single vehicle and eight vehicles) will not involve the LHN. The subsequent implementations that do involve a LHN have two choices that depend on the total number of vehicles and the number of LANs supporting the vehicles. Both of these choices assume that the velocity update algorithm will be used to allow 1/3-second updates over the network.

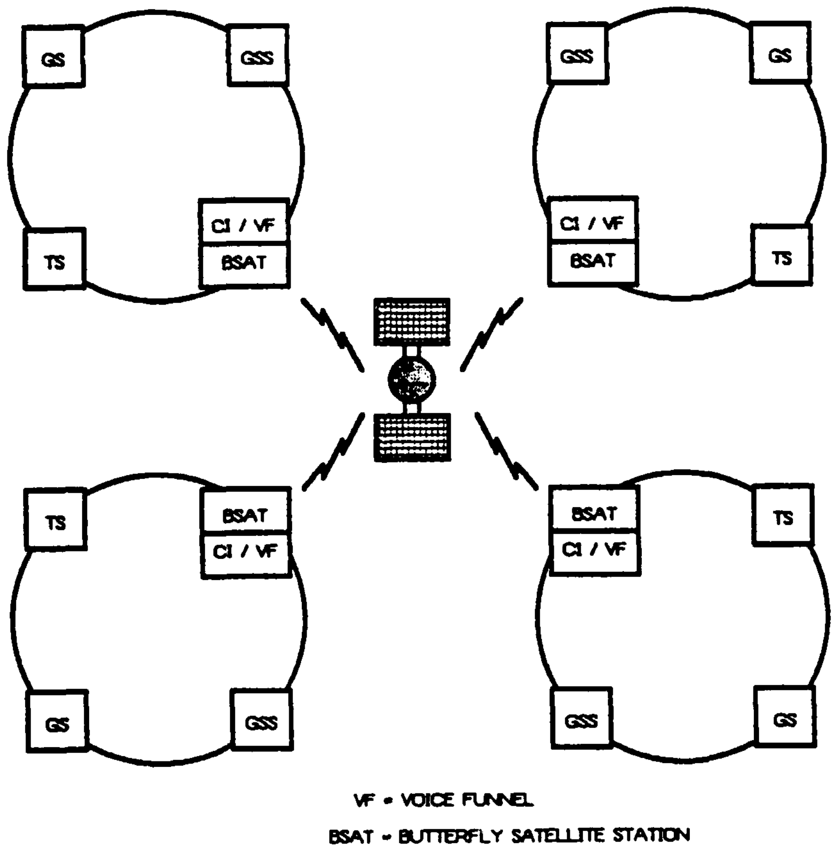
The first LHN solution is CIs connected by 56-KBit trunk lines. This alternative is viable when the total number of vehicles is under 100. Above this number, the state vector traffic starts saturating the trunkline bandwidth. The simplest hardware implementation of this is a CI consisting of a Multibus 68000 system with a LAN interface and an interface to a modem controlling a 56-KBit trunk line The function of the CI will be to receive packets from other CIs and put them on the LAN, while at the same time sending the local updates to all other CIs A packet switch would be overkill in this case because there is no routing involved. The most complicated task for the CI (other than handling the various device interfaces) is aggregating local updates into packet sizes that use the trunk lines more efficiently.

The second solution is a multiprocessor-based CI connected to a Satellite Broadcast network [Falk et al., 1981] (Figure 11). This is an ideal solution because most of the LHN is broadcast state vector updates. In addition, it can handle a much larger total number of vehicles (2000) than the previous solution In the course of investigating several LHN alternatives, we discovered that the Large Scale Simulation Network application s network requirements were very similar to those of the Voice Funnel project. Since it is already connected to the satellite broadcast network [Rettberg, 1982], We realized that a Voice Funnel processor could be used directly as a SimNet CI if we did two things.

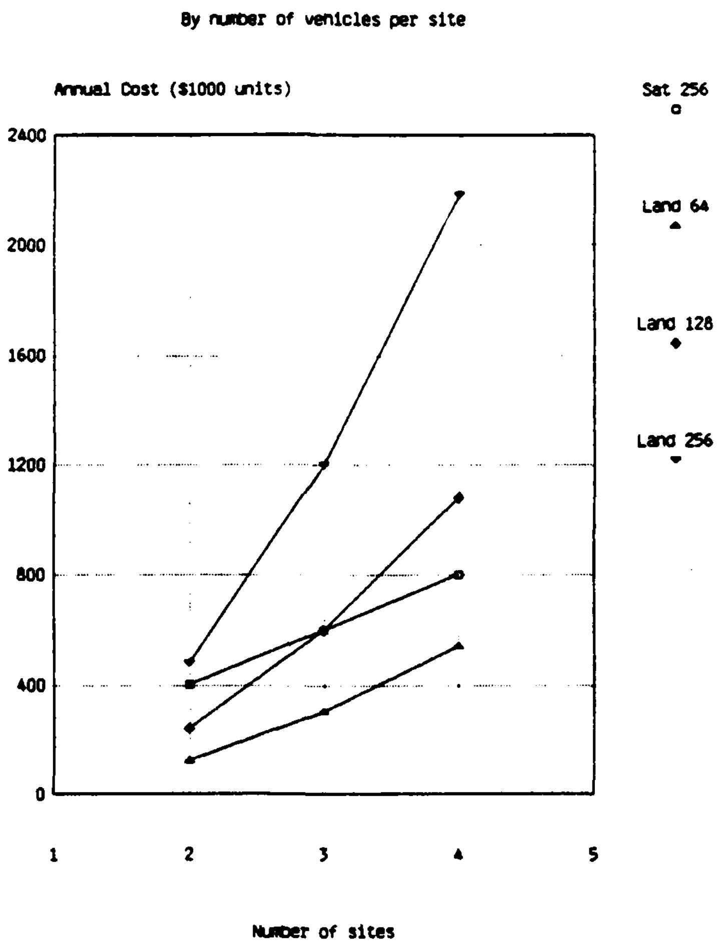
1. Connected the Voice Funnel processor to the SimNet LAN.

2. Made state vector update traffic use the ST protocol that the Voice Funnel application used.

The remaining question to be answered is, what are the relative costs of the two LHNs? The answer is dependent on the configuration. The following graph shows the relationship between cost and number of sites, comparing land line solutions against satellite for 64, 128, and 256 vehicles per site (Figure 12). The satellite annual costs are estimated at $200,000.



Satellite Network Configuration
Figure 11



Land Line vs. Satellite Cost Comparison
Figure 12

6.4.3 Network Reliability and Fault Tolerance

An important question to ask of the overall SimNet design is. What happens if messages get lost? There are two major classes of failure that we will consider, network failure and individual message failure.

At this point we do not have access to hardware reliability estimates for the LANs under consideration. However, practical experience indicates that it is usually on the order of months between individual node failures on the LANs we have worked with. Total network failures are even rarer (except when making changes that affect the transmission medium, such as adding or removing nodes).

Since LHNs have more components that can fail, they do not have the reliability characteristics of LANs. In the case of the Arpanet, there are extensive performance statistics that give information on node and trunk line reliability. These statisics indicate that the mean time between failure for nodes is approximately 300 hours, and that trunk lines are rarely down more than 2 percent of the time [BBNCC, 1983].

Our experience tells us that the combination of LAN and LHN is reliable enough to allow at least tens of hours of SimNet session time without network failure. Unless there are reasons for not doing so, it seems prudent to have the ability to stop and save a session upon detection of network failure. The session could then be continued as soon as the network recovered. and the elapsed session time would not be wasted.

The SimNet design is much more tolerant of individual message failure than it is of network failure. Since the networks we are considering (both local area and long haul) are highly reliable when they are functioning properly, individual message failure for the system as whole will be relatively infrequent. Message failure for a particular transmitting vehicle will be much more infrequent than message failure for all vehicles. Since each vehicle is sending 3 updates a second, it is highly unlikely (barring network failure) that more than a second will go by without an update being received from a particular vehicle. With the velocity update algorithm, the display process can continue without any noticable bad effects in the event of an occasional missed update.

6.4.4 Development Plan for SimNet Network

The first phase o. the development plan for SimNet network is to integrate a single vehicle with a LAN. This step accomplishes two things. first, it gives efficient access to the simulation processor for development purposes (downloading. debugging, etc.), and second, it provides a network environment for debugging and testing the TS and GM functions. The first implementation of the LAN will be an Ethernet, because of its immediate availability and the existence of support software and tools that use it. The development simulation processor will be a Q-Bus 68000 processor with a Q-Bus Ethernet interface. The Ethernet will remain in place until after the single vehicle demonstration.

The second phase consists of switching from Ethernet to a ring network. First, only the device hardware and lowest level network drivers will be replaced. Then, as hardware and firmware optimizations are added, software implementing the SimNet network interface (described in Network Interface section) will be removed or modified as appropriate. At this point, the ring network architecture will be fully integrated into the SimNet LAN.

The third phase will begin with building prototype LHN capability. The prototype CI will be built that will directly connect two LANS locally. Initially, trunk line (or satellite) delay will be simulated in the CI. The CI application will be built and tested, and update rate and delay experiments will be performed with various vehicle and LAN configurations.

7 Simulation

The simulation is the last part of the SimNet design presented. The software and hardware aspects of the design are discussed in separate sections.

7.1 Simulation Software

The primary simulation function occurs in the GSs and the secondary simulation function occurs in the GSS. Let us consider simulation in the GS first, and then in the GSS.

7.1.1 Game Station Simulation Aspects

7.1.1.1 Overview

Each GS consists of a vehicle simulator console, a graphics processor, and a simulation processor. All the I/O devices of the vehicle are connected to the console, and are handled by the simulation processor with the help of the Interaction Device Controller (IDC) (see Appendix C), the exception being the color video displays, which are outputs for the graphics processor. The graphics processor displays the terrain, foreground texture points, tanks explosions, and muzzle flashes. The simulation processor performs the application functions move, shoot, communicate, and maintain. It also performs internal storage management of the terrain database, and requests terrain patch updates from the TS as needed. Information about the position of this vehicle and any rounds fired are made available to the graphics processor by the simulation processor, whereas the same information for other vehicles participating in the exercise is provided by the network interface. This interface is also the way GSs interact with the GSS. The simulation processor will therefore consist of the following major software modules:

o Multitasking operating system.

o IDC for handling the control elements of the simulator console.

- o Applications modules for move, shoot, communicate, and breakdown/resupply functions,
- o Terrain patch management,
- o GSS interface, and
- o Network communications interface.

Figure 13 shows the major software modules in the GS.

7.1.1.2 Simulation Details

Multitasking Operating System

The operating system residing in the simulation and graphics MC68000 processors is CMOS, a small, real-time, multitasking operating systems written in C [Burke and Stearn, 1981]. The other modules will be tasks under CMOS.

Interaction Device Controller (IDC)

The input devices will be read by the input processor part of IDC, which will determine whether some input has changed and will then notify the rest of the software about any changes. The output devices under the control of the IDC will be dials, lights, and the sound generator (color video displays excluded). The output processor of the IDC will handle all the format conversions before refreshing or writing the outputs. The protocol for output to an IDC is examined in detail in Appendix C. In Appendix B all tank dials and controls are listed.

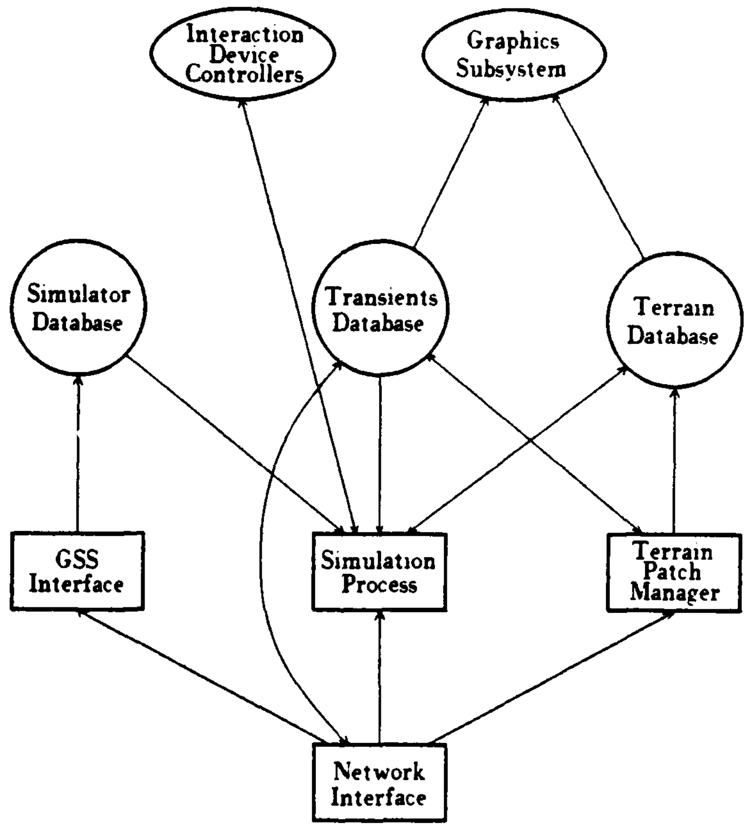


Figure 13. GS Software Modules

Applications Modules

The applications modules will handle not only effects but also constraints. Let us consider the effects first. In the case of the move function, the effects of slope, surface hardness, penetrability of objects, and engine noise proportional to RPM will be included. Likewise, the shoot function effects of turret traversal, gun elevation, magnification in the sights, lead tracking, rangefinding, ballistic computations, muzzle flash, hit effects (video and audio) will all be included. In the area of communications, capability to make calls through the chain-of-command for maintenance, indirect fire, and supplies will be enabled or disabled by the simulation processor. Finally, the maintenance/resupply function effects such as failing components, latency in resupply, or repair will all be part of the simulation.

Among constraints to be included in the simulation process will be those for position, speed, and acceleration for the move function. For example, a tank cannot be on a slope greater than a certain maximum, a tank cannot exceed the maximum speed for the terrain it is on; rate of acceleration from start and deceleration on braking cannot exceed their rated values. Similar examples of constraints on the shoot function are: the gun elevation is restricted to be within some fixed angles in the forward or rearward directions, the gun turret slew rate is fixed, the loader can only load a certain number of rounds per minute, etc. Examples of constraints to be included in the maintenance function are those of time taken for supplies to reach the vehicle, time taken to repair will be governed by the MTTR of the failed component, resupply will be possible only to the extent that requested supplies are available at the supply site, etc.

Terrain Patch Requests

One of the functions performed by the simulation processor in the GS is to ask the TS for the terrain patch (a 10km x 10km subset of the full 100km x 100km terrain) that corresponds to the vehicle's position on the terrain. This request would be a message of the type:

```
request_patch (station_id)(position)
```

This results in a terrain patch centered around "position" to be sent to the GS of id "station_id, by the TS.

Later on, as the vehicle moves and approaches the edge of its terrain patch, the GS will need to acquire new terrain data. The simulation processor determines this need and sends the following request to the TS.

```
update_patch (station_id)(old_position)(new_position)
```

This causes the terrain patch update, patch(new_position) - patch(old_position), to be sent from the TS to the GS of id "(station_id)".

The terrain patch manager in the simulation processor frees the space occupied by terrain data that is not being used any more to accommodate the terrain patch update.

GSS Interface

The GSS interface handles all transactions between the GS and the GSS. These activities include listening to exercise start/stop commands, receiving initialization data from the GSS and storing it in the simulator data base, and receiving time initialization.

Network Interface

The network distributes vehicle information from one GS to another. This information can be categorized into two classes, synchronous and asynchronous, depending upon how often they need to be sent. Synchronous information is sent repetitively at some frequency, the most important examples being vehicle position, orientation, etc. Asynchronous information is sent sporadically or in bursts. Terrain patch updates fall in this category. Information such as round fired (GSs respond to this by showing appropriate muzzle flash) or hit (GSs show hit effects) is really asynchronous, but is sent in the synchronous mode for reasons of convenience. The network interface interacts with the transients data base in the GS in the exchange of these types of information. Section 6.2.2 contains details of the data structure involved.

7.1.2 Game Support Station Simulation Aspects

In addition to the main simulation occurring at the GS, there will be other simulation functions performed by the GSS: control of maintenance/resupply vehicles, and control of indirect fire.

7.1.3 Control of Maintenance/Resupply Vehicles

Maintenance and resupply vehicles are operated by the GM. He receives a request for certain supplies over the voice communications system. He examines the local data base to determine the availability of that supply. If those supplies are available, then he again looks in the data base to see what type of vehicle is available to carry the supplies to the delivery site. He then dispatches that vehicle. The GSS applies checks to see that constraints, such as amount of supplies carried and the rate at which they can be delivered, are satisfied. A probabilistic model determines whether the supply vehicle ever reaches its destination, simulating the effects of enemy interception of supplies. If all goes well for the supply vehicle, an appropriate icon appears at the delivery site, after the appropriate amount of time. It will at that time be susceptible to hits by enemy fire. The GS that requested the supplies will be resupplied only after the lapse of a certain amount of time, which corresponds to the supply transfer time.

Maintenance vehicles are operated in a similar manner. Transit time from base to site will be appropriate for that vehicle, and mean-time-to-repair will also be observed, with suitable statistical fluctuations. As in the case of the supply vehicles, maintenance vehicles will also be subject to enemy interception.

7.1.4 Control of Indirect Fire

Indirect fire will be requested by GSs at a specified location, time, and duration. If any vehicle wanders into the indirect fire zone at this time, there is a high likelihood of it being hit. The GSS watches the locations of all the vehicles in the simulation and sends them hit messages as appropriate.

7.2 Simulation Processor Hardware

The simulation computer (SC) will perform all upper-level tasks involved in simulating the M1. This system will consist of two 68000 processors. One of these will be responsible for upper-level graphics database management and will directly control the graphics system. The second of these two processors will perform the actual vehicle simulation as well as managing network communications.

As stated before, estimates show that the graphics database management task will consume 66% to 75% of a 10MHz 68000. This leaves a sufficient margin of safety for tasks not considered in the estimates, but does not allow any cycles for other major tasks. For this reason, a second 68000 is required. This second 68000 will perform all of the vehicle dynamics simulation. This 68000 will also be receiving inputs from the interaction device controller. It will manage the state vectors received from the network as well as provide its own state to the network. Thus, all network functions will be assigned to this second 68000.

Each 68000 will require substantial amounts of local memory. The terrain database will occupy roughly .67MBytes of memory (See section 5.1.5). This implies that the graphics 68000 will need 1.5MBytes of local memory for code and data. The other 68000 must maintain all simulation variables and network buffers. Here the requirement is less, but certainly not less than .5MBytes, and will probably require 1MByte of local memory.

7.2.1 Solutions

7.2.1.1 Development Phase

During the development phase of the project two 68000s will be used. One will be supplied from Adage and is part of their standard 3000 product line. The second will be on a Q-Bus system and is also an "off-the-shelf" item. The Q-Bus approach is used because an Adage to Q-Bus interface is available and network interfaces are available for the Q-Bus. This interim solution will provide a mechanism to allow software development to progress while final hardware is being designed and implemented. See Figure 14 for a block diagram of this system.

This configuration does not well match the memory and configuration requirements of the final system. The Adage 68000 is more closely coupled to the Adage system, but does not have sufficient memory to maintain the terrain patch. Also it does not provide a mechanism for attaching a network interface. Therefore, in the development phase, the Q-Bus 68000 will deal with the network, simulation, and terrain data, while the Adage 68000 performs upper-level graphics tasks by caching data from the Q-Bus system.

7.2.1.2 Low Unit Cost Solution

The optimal solution is to design a set of boards that are directly installed into the Adage backplane. These boards would provide two 68000s, local memory, and a network interface. (See Figure 15 for a block diagram of such a system.)

In such a subsystem, one board would contain the Adage bus interface, the network interface, the two 68000s, and memory shared between the 68000s and network interface. Bulk memory required for code and large amounts of data would be on an additional board, which could be dedicated to one of the 68000s. With this scheme, a three-board system would provide the two 68000s, network interface, and sufficient memory for each of the processors.

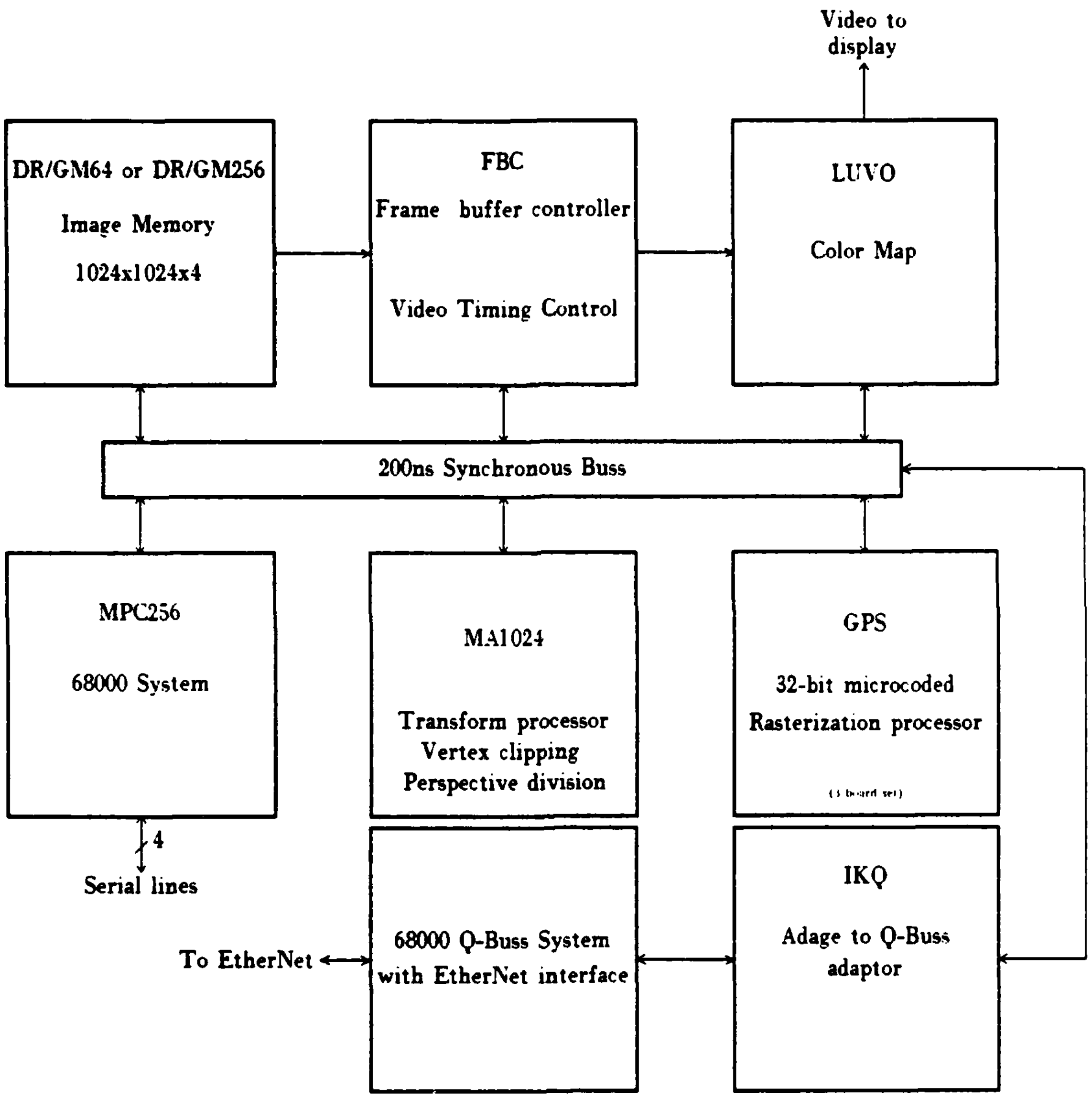


Figure 14. Interim Simulation Processor Block Diagram

This would provide a solution tailored to the SimNet task. No unneeded hardware would exist. However, the solution requires that two boards be designed, the dual processor with network interface board, and the bulk memory board. The memory board is

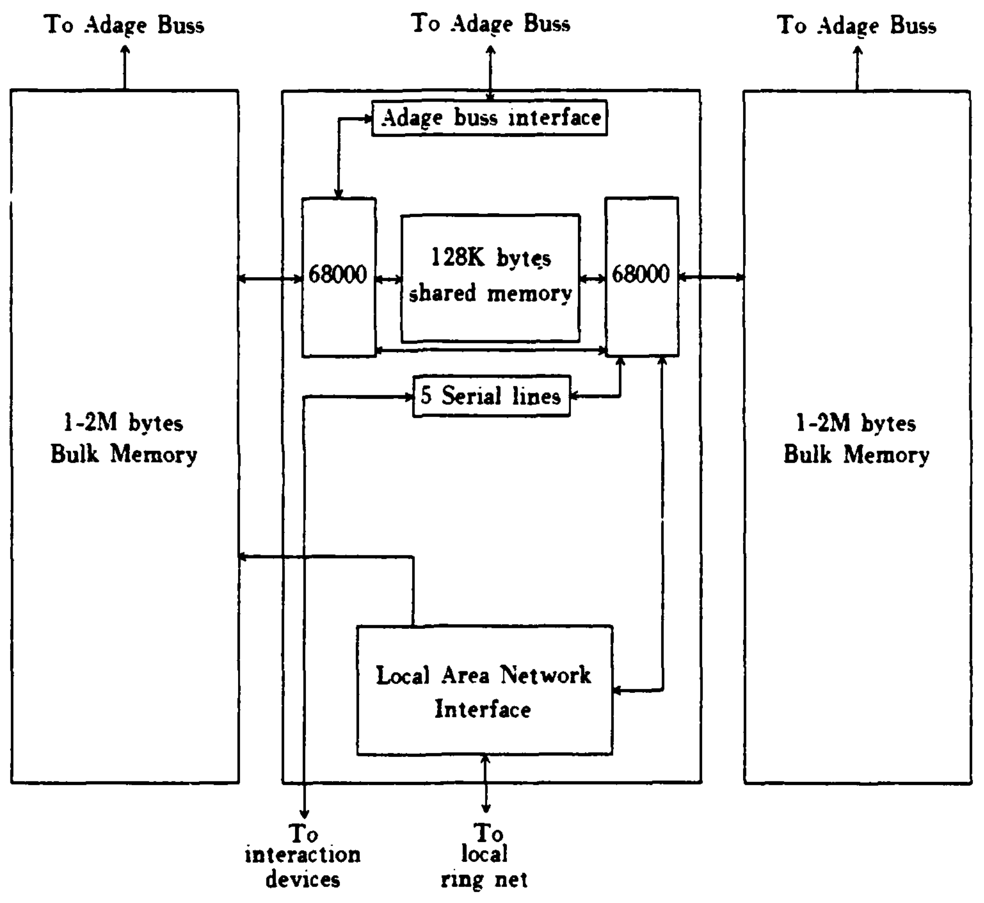


Figure 15. Simulation Processor Block Diagram

a straightforward design, but the dual processor requires some care, particularly in the network interface.

7.2.1.3 Low Development Solution

The 68000 system supplied by Adage has several problems. First is high unit cost. It can support at most 512 Kbytes of memory and has no way to interconnect an available network interface. For these reasons, it would be best if the final solution did not include the Adage 68000.

The Q-Bus 68000 has its own local memory bus, allowing the 68000 to run at 10MHz with no wait states. This also does not use any Q-Bus cycles. By putting two 68000 boards on a single Q-Bus, the system can have the required two processors, each with the required memory. Network interfaces are readily available for the Q-Bus.

Disadvantages to this solution are unit cost and Q-Bus bandwidth. The Q-Bus can support a burst transfer rate of 1 MByte per second. Periods of peak network activity may interfere with graphics data traffic to the Adage system. The cost will be more than doubled by using a Q-Bus system instead of a custom-designed board set for the simulation computer.

7.2.2 Conclusions

Designing a custom set of boards that install into the Adage backplane is the best solution to the SC. It offers the lowest cost per copy and the best hardware fit to the problem. The per-copy price for the custom solution will be roughly half that of the Q-Bus or Q-Bus/Adage solution.

7.2.3 Hardware Design

7.2.3.1 Low Unit Cost Solution

This solution is envisioned as a three-board set. One board contains two 68000s, a network interface, serial lines to the IDCs, and some shared memory. The other two boards are memory and are identical. These provide independent bulk memory for each of the two 68000s.

The graphics 68000 must run primarily at 10MHz This is required to perform the various graphics database management tasks. Some wait states may be permissible for actions that occur relatively infrequently This processor also requires about .67Mbytes of memory for the local terrain patch Additional memory is required for the various intermediate display lists generated by the sorts performed by the 68000 These display lists must then be passed to the AGG4/MA1024 for transform processing. This must occur with very little overhead to the graphics 68000. In addition, new terrain data from the network, as well as updated tank state vectors, must be merged into the terrain data. This must also take place with little overhead.

The simulation 68000 will be responsible for the network, IDCs, and physical simulation. The network interface must be able to handle back-to-back tank updates and not miss them This should also occur in a manner that does not totally utilize the simualtion 68000, which requires some intelligence in the network interface. The simulation 68000 must be able to interrogate the terrain data periodically to determine the slope of the current M1 position. This can be performed by directly inspecting the terrain data, or by interaction with the graphics 68000. Target objects must also be extracted in a similar manner These tasks require a communication path between the two 68000s.

The development effort for this subsystem will require 4.5 person-months for the processor and 2.5 for the memory. Each board will require six weeks for prototype board fabrication. Prototype checkout will take one month for the processor and three weeks for the memory.

7.2.3.2 Low Development Solution

No hardware development is required for this solution. All components are off-the-shelf items. An evalution must be made to choose which configuration provides the best match to the requirements of the SC.

Either a straight Q-Bus or a Q-Bus/Adage system could be used. The Q-Bus may not be fast enough (1 MByte per second in burst mode) to handle the network traffic and the graphics data.

The Adage 68000 has a limited amount of memory available (512 Kb), which restricts the set of tasks it can perform.

PART IV. SUMMARY AND CONCLUSIONS

The preceding sections of this report have been a description of the functions and the technical design of SimNet. Because SimNet is a project involved with demonstrating a new training technology, it is obvious that the resulting implementation will not be the last word on the subject. Rather, it is an attempt to introduce new methods of using computer graphics, simulation, and networking to the problems of tactical training of groups of people. Thus, there will be much to be learned about the effectiveness of the ideas for training, as well as the effectiveness of the implementation approaches described here in conveying those ideas. The purpose of this part of the report is to try to look to the future of this approach by suggesting areas for improvement on this design, directions for making use of the technology, and estimates of how it scales to larger problems and problems in other training domains.

8 Future Directions

There are three areas for future development of the ideas exemplified by the SimNet prototype trainer: improvements to and enhancements of the prototype's functionality, scaling the technology to accommodate larger numbers of participants, and development of generic team training concepts that can be transferred to other domains. The first two, enhancements to functionality and scaling, are of direct applicability to the development of a production device that can be introduced into military training programs. The last area, generic concepts, might have impact on a much wider range of problems. It is this area that the SimNet program is ultimately concerned with.

8.1 Enhancements of the Current Design

There are a number of effects that have been excluded in the initial design. These exclusions, which are discussed in section 2.3, were chosen to spe*d development of the prototype and to reduce its cost and complexity. Effects such as smoke, night fighting, mines, destruction of terrain and cultural features, and additional combined arms elements such as air strikes, are obvious enhancements to the basic functionality of the system. Such effects are consistent with the overall SimNet design and could probably be introduced with little fundamental change.

Some enhancements, however, are not as straightforward to introduce into the system, and would entail major redesign or additions that alter some of the fundamental design principles. For example, the introduction of infantry or massed groups of forces controlled by a single human participant entails "role playing" that is not part of the current design. Even more ambitious would be the introduction of totally automated simulation of participants, either as individuals or massed forces. These involve the development or adaptation of sophisticated computer simulation models that are far beyond the dynamics simulation around which the current design is based. These capabilities are important, however, when attempting to scale this type of system to involve very large numbers of participants.

The introduction of true OPFOR tactics into the SimNet environment is another possible enhancement of the system. The problems this enhancement presents are more a matter of policy than technology. Certainly, the simulation of Soviet tank dynamics fits easily within the current SimNet framework. The introduction of Soviet crew tactics, however, requires either the type of sophisticated computer simulation mentioned above or the use of specially trained personnel who play the role of OPFOR troops, as is done in the National Training Center.

The training environment in which SimNet is used represents another area for development. The current system does not incorporate such aids as data collection for "instant replay," after action review, or "what if" scenarios. Each of these functions is important in a complete training environment, and the adaptation of the system to such an environment is a matter for further study.

A final area for enhancement of the current SimNet implementation is the application of new hardware technology to lower the cost of individual components such as the GS. The graphics aspects of the design are especially amenable to this type of development. For example, the introduction of VLSI multiprocessors for display generation, and smaller, cheaper bulk memory will go a long way toward achieving the goal of high-performance, low-cost SimNet stations.

8.2 Very Large Scale Gaming

The current goal of the SimNet program is to develop multi-player trainers for hundreds of participants. A longer term goal is to understand how this team training technology will scale to accommodate thousands of players in single or, more likely, multiple concurrent exercises. Larger systems allow not only the training of large numbers of personnel in a potentially more cost-effective manner than traditional methods, but can also approach the problem of training commanders of large forces, for which few mechanisms currently exist.

The problems of extending the current ideas to allow several orders of magnitude more participants are both technological (i.e., hardware- and software-related) and practical. For example, one problem is coordination of the activity of thousands of players in using such a system. Scheduling a single exercise with so many players is a tremendous logistical problem and is potentially wasteful of resources. One solution, as discussed above, might be the use of computer simulation to play the role of large numbers of individual or massed forces. Another technique that might make such large systems workable is to allow participants to join one of many ongoing exercises. A number of long-term exercises might be in progress simultaneously, with players coming and going in some coordinated fashion. (This is analogous to "telegaming," which is becoming popular on computer information services like Computrend, where owners of personal computers can dial up the service and join an ongoing "Adventure"-style game.)

The more apparent technical problems with very large scale simulation are centered around the communications area. As discussed in Section 6, factors such as delay and bandwidth, which are manageable for hundreds of stations using current networking technology, soon become overwhelming with more than a few hundred stations. While networks of 64 stations require a LHN bandwidth of about 50Kbps, which is within the capability of current long haul packet switch technology (e.g., Arpanet) larger networks of just 256 stations require 200Kbps, which exceeds current terrestrial long haul technology. The problems of delay in the LHN are even worse for larger networks, approaching one second for 256 stations using Arpanet technology.

The solutions to these problems lie in several areas. First, wideband satellite technology offers greater bandwidths and more constant delay characteristics over long distances. However, even satellite channel bandwidths of 1-3Mbps may be insufficient for thousands of stations using the update algorithms that have been planned for SimNet. The analysis of Section 6.2.5 suggests an upper bound of 2000 stations using wideband satellite technology. Alternatively, bandwidth and delay requirements can be reduced by techniques such as using incremental algorithms that broadcast changes in speed and acceleration beyond some threshold, rather than absolute positions, reducing the need for frequent updates. Another possibility is to divide large exercises into segregated "channels" that contain concurrent sub-exercises.

These are just a few of the major problem areas involved in attempting to scale SimNet technology to much larger networks. This area is ripe for further research and can naturally build on the results of SimNet development.

8.3 Generic Team Training Concepts

While the primary focus of the SimNet project is the delivery of a prototype training system, it is important to realize that an important goal of the program is the development of a new training technology. Thus, while a trainer centered around an M1 tank simulator is a useful exemplar for proving the feasibility of distributed, multi-player gaming, it is also an experimental vehicle for developing generic concepts of team training. Even at this early stage of the program, we can identify concepts that will have applicability to other domains, both within the military (air or naval combat) and outside it (international relations, economics). For example, communications issues, such as reduction of bandwidth and delay demands by the use of robust and compact data distribution and update algorithms, are immediately suggested by our work with the current SimNet design.

There are a number of areas where we can abstract concepts of general applicability in team training. More attention must be paid to the definition of _training goals_ of the technology. Defining what a system is trying to train is important. Team training as exemplified by SimNet may be effective (both in cost and purpose) for training large groups, coordinated arms training, or training of commanders, since these are labor-intensive tasks that may not be possible by other methods. Alternatively, hand/eye coordination is probably more effectively taught with specialized individual trainers.

The _training environment_ in which such team approaches are to be used must also be well defined and appropriate to the task. For example, the way SimNet is integrated into the chain of command attempts to mimic the actual communications paths between echelons. For other applications, it may be appropriate to give commanders more information than is available in an actual battlefield situation to allow more active intervention into troop training activities.

In the technology implementation areas (simulation, networking, and graphics), there are other generic concepts to be developed. For example, the use of sophisticated "_role playing_" _simulations_ that can replace individuals or groups of participants has already been discussed as a technique for extending team training to larger domains. In networking, the development of _generic simulation protocols_ that reduce bandwidth and delay requirments and increase robustness in the face of lost messages is another area we have already discussed. Other generic communications issues include the use of real-time _teleconferencing_ to provide a relatively unstructured environment for assisting in training (evaluation, etc.), _adaptation of existing long haul communications networks_, such as Arpanet, MILNET, and commercial X.25 networks, for use by team training applications to reduce communications installation and development costs; and the use of _multiplexing_ concurrent training exercises on a single system, as mentioned above. Finally, in the graphics area, generally applicable techniques include the exploration of different modes of presentation of simulation activity, such as realistic perspective views versus planar schematic views (e.g., maps), and the development of low-cost display technology that can be applied to make team trainers cheaper.

As part of our work in the SimNet project, we will begin to explore some of these generic approaches and will report on the results of these explorations in future documents.

9 Conclusion

We have attempted in this report to present our design for a prototype SimNet trainer. As we have seen, this involves much more than just the details of the hardware and software implementation of the system. It also concerns the definition of functions of the trainer and the environment it is intended for. In addition, we have provided an introduction to the more general topics of team training being developed in the SimNet program, which are areas for future research.

We have tried to present as complete a picture of this project as we could after only six months of effort. Naturally, other details have been omitted that will need to be presented in later reports. Also, as is the case with all engineering designs under development, the details are constantly evolving, and many issues will have been understood in better detail, changed, or discovered by the time this report is published.

It is important, by way of conclusion, that we put the goals of this project in perspective. We have stated several times in this report that we are building a prototype trainer and not a production system. One implication of this is that we have also not built a complete training environment with evaluation facilities for individual users, which are normally part of a trainer. A more important implication is that the ultimate effectiveness of the prototype or, indeed, of the team training approach itself, is unknown. Therefore, it is imperative that the resulting system undergo extensive evaluation, both in terms of its effectiveness as a trainer and in terms of the general effectiveness of distributed multi-player gaming as a method of training. Since the broader goal of the SimNet program is to develop a new training technology, this type of evaluation, as well as the articulation of the general concepts that have been developed, is crucial to its success.

10 References

[BBNCC, 1983] BBN Communications Corporation, ARPANET Summary Data for the month of August 1983.

[Bloedorn, et al., 1983] Bloedorn, G., Kaplan, R., and Jacobs, R., Large Scale Simulation Data Package, Perceptronics, Woodland Hills, CA, 1983

[Burke and Stearn, 1981] Burke, E. and Stearn, J., "CMOS Overview Description and Specification," BBN internal documentation, 1981.

[Bux, 1981] Bux, W., "Local Area Subnetworks. A Performance Comparison," Local Networks for Computer Communication, North-Holland Publishing Co., Amsterdam, 1981.

[Falk, et al., 1981] Falk, G., Groff, S., Koolish, R., and Milliken, W., "PSAT Technical Report," BBN Report No. 4469, May 1981.

[Gurwitz, 1983] Gurwitz, R., "LSG Notes #2 - Graphics Hardware Comparison," June 1983.

[Rettberg, 1982] Rettberg, R., "Development of a Voice Funnel System," BBN Report No. 4928, March 1982.

APPENDIX

11 Appendix A. Typical SimNet Exercise

Three aspects of an exercise are considered in this appendix. exercise preparations. initialization. and the exercise itself.

11.1 Exercise Preparations

There are two exercise preparation steps. the one-time preparation of generic files that contain information which can be used from exercise to exercise. and the recurring preparation of files that are specific to an exercise and therefore have to be changed from one exercise to another.

11.1.1 One-time Preparations

There are two different activities involved, one involving the preparation of generic files, and the other using one of these files to produce a specific terrain.

First, let us consider the preparation of the terrain templates file (TTfile), the vehicles file (Vfile). and the ordnance file (Ofile). The TTfile contains templates of objects found in any terrain, such as hills. rivers. roads. trees. etc. The templates require parameters such as location. height. width. etc.. to be supplied. The Vfile contains detailed information about each type of vehicle that could possibly participate in the exercise. Finally. the Ofile contains data about the different types of ordnance that could be used in the exercise.

Second, let us build ourselves a specific terrain. Building a specific terrain essentially consists of making a list of all the objects in the terrain, choosing the terrain template appropriate for each one, providing all the arguments for each template (location, height, width, etc.), and finally, saving everything in a file called the terrain specifications file (TSfile). The terrain tool merges the TSfile with the TTFile to produce an actual hard-copy map of the terrain (to be distributed to the crews) as well as a machine-readable copy of the same map.

The second activity is time consuming, and should be done well in advance of the exercise.

11.1.2 Recurring Preparations

This puts specific vehicles on the terrain built above.

This activity involves placing vehicles at specific locations on the terrain, carrying specified loads of fuel and ordnance. These descriptions are stored in the exercise Layout file (GLfile). This file is then processed along with the Vfile and the Ofile by the exercise processing tool. The result of this process is the exercise File (Gfile). A special note at this point: the GSS also operates supply and maintenance vehicles, whose parameters are stored just as those for the other vehicles are.

11.2 Initialization

The crews are given copies of the Terrain Map.

All the GSs, the TS, and the GSS are powered on. Then they each run their own diagnostics/self-test programs. After this, these components go into a "wait" (listen) state.

The GM at the GSS determines that all components are in working order. If they are not, there are three alternatives: first, ignore the component (this alternative will work for GSs, but not for the TS); second, have the component repaired and then proceed with the exercise; third, prepare a new Gfile.

Having confirmed that all components are ready, the-GM selects the appropriate TMfile and Gfile and loads the components of SimNet with their respective data bases. The components receive this information and again go into the wait (listen) state.

Now the GM establishes the exercise time by means of the "Set Time" command.

Finally, the GM gives the "Start exercise" command from his console.

11.3 Introduction to Detailed Exercise Scenario

The purpose of training is to increase a unit's proficiency in accomplishing its missions. All training is therefore mission-oriented. Hands-on, performance-oriented training is considered the most effective means to increase the efficiency and proficiency of personnel ([Bloedorn et al., 1983], p13-27)

Since training is mission-oriented, the following sample training sessions are identified by the type of mission that is the subject of the training. Only two examples are given here.

Move

Conduct Fire and Maneuver

Each scenario simulates a platoon-level drill, however, some attention is also paid to the activities and experiences of personnel at the individual tank level. The following descriptions are from the viewpoint of both the platoon leader (PL) and the personnel of a given tank in the PL's command.

11.4 Move Drill

11.4.1 Training Objective

The training objective for this drill is to move as a platoon observing the following standards:*

- o The platoon must reach the start point (SP) and the release point (RP) at the predesignated times.
- o The platoon must move along a covered and concealed route.
- o The platoon must move in a staggered column.
- o Individual vehicles must maintain 50 to 100 meter intervals.
- o Each vehicle must keep weapons oriented over its assigned primary area of surveillance responsibility.
- o All OPFOR targets must be acquired as they are identified.
- o Upon achieving the release point, the platoon must form up in a coil or herringbone formation.

11.4.2 The Drill

The platoon leader (PL) receives operations orders to move his platoon from a given SP to a specified RP according to a specific timetable.**

1. The PL selects a movement route using a map and determines the platoon's movement technique (selecting from those applicable when not in contact with the enemy). Using radio communication, the PL issues instructions to his subordinate tank commanders

* These standards have been directly derived from [Bloedorn et al., 1983], p. 13-3.
** This drill has been derived directly from [Bloedorn et al., 1983], pp. 13-32.3.

2. The tank commander (TC) selects an exact terrain route by analyzing terrain using the five military aspects of terrain maximizing cover and concealment. The TC orients his map and determines current location by map-terrain association.

3. While in motion, the TC navigates his vehicle by identifying terrain features on the map and directing the driver by means of intercom communications. The TC also maintains tactical communications with the platoon leader.

4. The driver negotiates the route (cross-country, road, etc.) indicated by the TC.

5. The gunner keeps the vehicle's weapons oriented on the assigned primary area of surveillance responsibility in accordance with standard operating procedure. Both the gunner and loader observe the surrounding terrain for indications of OPFOR forces and positions.

6. Upon achieving the RP, the PL directs the deployment of his platoon in proper formation and awaits further orders.

11.4.3 The SimNet Session

Platoon personnel occupy the four tank simulator stations designated for the training session. Each trainee takes his position in his respective simulated tank. Terrain maps are obtained and examined.

1. After completing his tactical planning, the PL transmits orders to his TCs, using radio communications, specifying the movement route and the technique of movement.

2. Each TC then performs his own tactical preparation, selecting an exact terrain route. The route selected is mainly cross-country, taking the tank through an area heavily covered with bushy undergrowth, providing plenty of cover with only one really exposed spot on the route -- a ford at a stream. Using his simulated periscope viewscreens, the TC orients his terrain map and determines

his current position.

3. Once the PL gives the order to move out, the TC maintains intercom communication with his personnel, directing the driver over the terrain route and monitoring reports from the gunner and loader as they perform surveillance. He notes the vehicle's progress by identifying terrain features through his simulated periscopes and associating them with features on his map. The foreground points highlighting the surrounding terrain in the simulated video display assist the TC in assessing terrain slope and texture.

4. The tank crosses the line of departure, maintaining its appointed position in the wedge formation. The driver manipulates his controls to conduct his vehicle along the route designated by the TC. The first portion of the route is along a road, covered on both sides by high trees. After two kilometers, the tank leaves the road and breaks into the bushes, angling toward the desired RP. The tank engine whine increases as the driver opens up the throttle to crash through the brush. The tank slows, since the cross-country terrain impedes the tank's normal road cruising speed. Maneuvering around trees and large rocks, the driver pursues the path instructed by the TC. Six kilometers into the route, the tank emerges from the surrounding brush, approaching the ford in the stream. As directed by the PL, the tanks move through the ford in pairs, with the stationary pair providing overwatch support for the advancing vehicles. The forward tank driver sends his vehicle lurching down to the stream's edge and makes his way across the water. Upon taking up a protected position on the other side, the driver pauses to perform overwatch for the pair of tanks advancing behind. The platoon resumes a wedge formation and proceeds to the RP.

5. The gunner keeps the turret trained over the tank's primary area of surveillance responsibility and carefully watches for traces of OPFOR positions. The loader, assisting the gunner in his surveillance duties, spots dust billowing 2 kilometers distant. He notifies the TC, knowing that this could indicate one or more OPFOR vehicles headed in their direction. The gunner swings the turret around to acquire the possible OPFOR target and to determine the vehicle's

speed and heading. Meanwhile the TC notifies the PL of the sighting. The PL verifies the sighting as friendly vehicles proceeding on similar orders to the same RP. The driver and loader resume their standard surveillance posture.

6. Upon reaching the RP, the PL directs the deployment of his tanks in a coil formation. He instructs the TCs to perform resource accounting and anv necessary maintenance and awaits further orders regarding the rendezvous with the incoming sister tank platoon.

11.5 Conduct Fire and Maneuver Drill

The platoon must fire and maneuver as two elements, each a two-tank section, in keeping with Army Standard Operating Procedure (SOP). While one element advances, the other performs overwatch to protect the movement element. If necessary, the sections can then exchange roles, permitting a leapfrogging guarded advance.

The overwatch element must occupy a concealed, covered, and well-protected position that permits observation and fire over the terrain that the movement element will traverse. Hull-defilade positions are assumed when possible to maximize vehicle protection.

Overwatch vehicles must maintain 360-degree security for moving vehicles, detecting all OPFOR elements and placing fire on targets within fifteen (15) seconds of detection. Supporting fire (indirect artillery) may be requested and adjusted as required.

The movement element must move rapidly along a covered, concealed route to the next overwatch position. Where possible, movement element vehicles should employ fire while advancing. Upon reaching the overwatch position, the movement element deploys in concealed, covered positions, places fire on OPFOR positions, and performs overwatch for the advancing former overwatch element.

11.5.1 Training Objective

The objective of this drill is to provide practice in perfecting skills required by platoon fire and movement missions, observing the following standards.*

The platoon must maneuver and fire, engaging OPFOR armor until one or the other is neutralized.

The platoon must advance using standard platoon movement techniques applicable while in contact with the enemy.

Actions of the individual tanks of the platoon must be closely coordinated and controlled by the platoon leader (PL). The PL must specify target priorities and sectors of observation and fire, while maintaining accountability of personnel (status report, casualty report, etc.).

The PL should optimize the tactical effectiveness of his employment of available indirect fire.

OPFOR armored vehicles must be quickly identified and neutralized, making most effective use of available firepower. Coordination of individual tank internal activities as well as section and platoon cooperation are vital.

11.5.2 The Drill

The platoon leader (PL) receives operations orders to advance and neutralize OPFOR armor identified by friendly observers.**

1. The PL designates initial base of fire (overwatch) and movement elements and specifies subsequent overwatch positions. The PL also specifies the movement techniques

* These standards have been directly derived from [Bloedorn et al., 1983], pp. 13-34.5.
** This drill has been derived directly from [Bloedorn et al., 1983], pp. 13-34.5

to be used The PL must monitor the progress and status of all tanks in his platoon, coordinating their actions during the course of the assault. He must establish sectors of observation and fire, designating target priorities. Indirect fire requests must be placed when deemed necessary.

2. Each tank obeys orders issued by the PL, moving or providing overwatch as instructed. The TC must select exact routes for the tank and direct the driver over the route, reacting to both OPFOR direct and indirect fire.

3. The driver is responsible for conducting his tank through maneuvers directed by his TC, negotiating the existing terrain.

4. The gunner uses his Gunner's Primary Sight (GPS) to acquire targets as they present themselves and as directed by the TC. The gunner and loader together operate the main gun and coaxial machine gun to engage and neutralize OPFOR forces.

5. The drill terminates upon mission completion or when unable to proceed

11.5.3 The SimNet Session

Orders are received. The enemy has taken up positions controlling a major intersection. OPFOR armor must be engaged and neutralized.

The PL specifies the initial movement and overwatch elements (henceforth, Alpha and Bravo sections, respectively). The objective coordinates indicate that the target is within visual range but the OPFOR positions are concealed by trees and bush The PL requests supporting artillery on the target coordinates specified in his orders to provide additional covering fire until the exact enemy armor locations can be determined.

The indirect fire mission begins, but the PL sees on his monitors that the impact clouds are slightly off target. The PL transmits a grid coordinate shift adjustment to the artillery unit, and watches as the artillery responds and begins to register the target area.

The PL gives the order to move out, and Bravo section advances under the protective overwatch of Alpha section. Bravo section advances to the point indicated by the PL and assumes overwatch for the now moving Alpha section. Suddenly, the Bravo section leader (Platoon Sergeant/Tank Commander) sees two dust clouds moving in the target area. Apparently, the enemy armor is attempting to escape the pounding of the artillery fire by retreating from the intersection. He relays this by radio to the PL and orders his gunner and loader to prepare to fire a HEAT round at the nearer target cloud while directing the other Bravo tank to fire on the other dust cloud.

The gunner swings the turret around, ranges the target using the laser range finder, and positions the fire control reticle on the target. The gunner announces that the target is "identified," and the PS/TC confirms this by looking into his own view of the GPS.

Meanwhile, the loader swivels his chair to hit the knee switch that accesses the store 22 "ready" rounds. Each simulated round consists of two lights indicating HEAT or Sabot and a button that activates the simulation of removing that round. The loader examines this array of lights and finds a HEAT round (HEAT light on) and punches its button. This results in a delay that reflects the time required to remove the round and swing it around into the breech. During this delay the loader swivels his chair back to face the simulated breech of the main gun. After the two-second delay, the loader hits the arm button, simulating arming procedure. He announces "up" and assumes position for firing.

Finally the PS/TC notifies his PL and gives the order to fire. The round is away and the lack of a visible blast indicates that the shot did not destroy the target. Meanwhile the PS/TC receives a radio message from his sister Bravo tank that it fired and observed no blast, indicating indeterminate effects on the target.

The PL, moving up in Alpha section, orders continuing fire on the sightings and calls in a ceasefire request to the artillery unit to prevent the platoon from being hit by its own indirect fire. Alpha section takes up an overwatch position and Bravo section now begins advancing. The PL cautions his TCs to maintain observation for other OPFOR armor not yet sighted

Directing Bravo, the PS/TC orders his gunner and loader to fire again while on the move. Before the round can be fired, the PL's warning is justified. A loud blast is heard, signalling a hit registered on the tank. The PS/TC sees a muzzle flash from a concealed position just off the intersection. Not all the tanks retreated, perhaps as a ruse. Malfunction indicators light up, and the damage is serious. The main drive has been damaged, with the tracks possibly blown away. However, no one was injured and the tank's armament remains intact, so the PS/TC reports his status and the enemy position, orders the gunner to acquire the new target, and orders his driver to attempt clear the malfunction.

The PL monitors the reports of his TCs. The platoon is facing a full enemy tank platoon and two of his tanks have been hit, only one seriously. He directs both sections to concentrate fire on the new hidden OPFOR position, identifying it as the more immediate threat.

The PS/TC confirms his gunner's targeting and gives the order to fire, since a round was already loaded when the tank was hit. The result is a visible impact blast, the target tank is completely destroyed. The PS/TC notifies the PL and is awaiting orders when a loud blast comes over the sound effect speakers. The controls are dead. The tank has been destroyed.

The PL notes the loss and evaluates the situation. The two enemy tanks in the nearer position have been destroyed and the more distant tanks continue their retreat, deciding that their cause is hopeless. The PL orders his units to take up defensive positions around the intersection and to prepare for a possible enemy counter-attack. The TCs comply and transmit personnel and vehicle status reports to the PL. The PL transmits a request for maintenance/salvage for the destroyed tank and awaits further orders.

Eventually, the GM stops the exercise by means of the "Stop exercise" command.

12 Appendix B. Dials and Controls

This subject will be discussed in two parts The first part contains a brief description of the important M1 controls, the second part contains a detailed list of all the dials and controls of the M1 that will be replicated in the GS of SimNet

12.1 Brief Description

The important controls at the driver's station are the parking brake, the engine off-on switch, the steering bar with throttle, the transmission select knob, and the service brake, all of which are very much like those of a car with an automatic transmission.

The important controls at the gunner's station are the gunner's primary sight, including 3X and 10X magnification, the ballistic computer control panel, the turret traverse controls, the gun elevation controls, the laser rangefinder controls, the main gun (105 mm), the gunner's auxiliary sight and its controls, and the gun and ammunition select switches.

The important controls at the tank commander's station are the the turret traverse controls, the gun elevation controls, laser range finder controls, the triggers for main gun and coax machinegun, and the commander's gunner's primary sight extension

The important controls at the loader's station are the main gun breech operating handle, the knee switch to operate the ready ammunition bustle door, and the forty-four sets of lights and switches that represent the ammunition, each set containing two lights and a switch.

In addition to these controls, each tank crew member has an intercom attached to his helmet for voice communications within the tank. Additionally, the tank commander, gunner, and loader have the ability to establish radio communication with the outside world, including headquarters, command posts, and other vehicles.

12.2 Full List of M1 Controls and Dials

Commander - Optimum System

- 3 Unity Periscopes
- Frequency Selector Control
- Intercommunication Control
- Commander's Panel
  - Ready/Safe Switch
  - Fire Control Malfunction Light
  - Engine Fire Light
  - Manual Range Battlesight PB
  - Manual Range Add-Drop Switch
- Commander's Power Control Handle
  - Laser Rangefinder Button
  - Trigger
  - Palm Switch
- Commander's GPS Extension Eyepiece
- Commander's Remote Intercom Switch

Commander - Minimum System

- 3 Unity Periscopes
- Intercommunication Control
- Commander's Power Control Handle
  - Laser Rangefinder Button
  - Trigger
  - Palm Switch
- Commander's GPS Extension Eyepiece

Loader's Station - Optimum System

- Intercom Control
- Receiver R-422 VRC
- Receiver Transmitter
- Main Gun Breech Operating Handle
- Main Gun Status Lights
- Turret Blower Switch
- Knee Switch
- Periscope
- Audio Freq Amplifier

Loader's Station - Minimum System

- Intercom Control
- Receiver R-422 VRC
- Receiver Transmitter
- Main Gun Breech Operating Handle
- Main Gun Status Lights
- Turret Blower Switch
- Knee Switch
- Periscope
- Audio Freq Amplifier

Gunner's Station - Optimum System

- Gunner's Primary Sight
- Intercom Control
- Computer Control Panel
    - Test Key
    - No/Go Light
    - Battle Sight Adjust Key
    - On-Off Switch
    - Power Light
    - Range Key
    - Lead Key
    - Cant Key
    - Crosswind Key
    - Computer Display
    - Number Keys
        - 0-9
        - decimal
        - Dash (minus)
    - Enter Key
    - Clear Key
- Gunner's Power Control Handle
    - Palm Switches
    - Laser Range Buttons
    - Triggers
- Gunner's Foot Intercom Switch
- Unity Periscopes
- Fire Control Mode Lights
    - Normal
    - Emergency
- Fire Control Mode Switch
- Gun Select Lights
    - Main
    - Safe

- Coax
- Gun Select Switch
- Ammo Select Switch
- Ammo Select Lights
  - HEAT
  - APDS
  - HEP
  - BH
- GPS Magnification Lever
- Gunner's GPS Eyepiece
- Range Switch
- Thermal Magnification Lever
- (TIS) Polarity Switch
- (TIS) TRV Ready Light
- (TIS) Fault Light
- (TIS) Thermal Mode Switch

Gunner's Station - Minimum System

- Gunner's Primary Sight
- Intercom Control
- Gunner's Power Control Handle
  - Palm Switches
  - Laser Range Buttons
  - Triggers
- Unity Periscopes
- Gun Select Lights
  - Main
  - Safe
  - Coax
- Gun Select Switch
- Ammo Select Switch
- Ammo Select Lights
  - HEAT
  - APDS
  - HEP
  - BH
- GPS Magnification Lever
- Gunner's GPS Eyepiece
- Range Switch

Driver's Station - Optimum System

- Alert Panel

- Master Warning Light
- Reset Pushbutton
- Master Caution Light
- Steer-Throttle Control
  - Steering Bar
  - Twist Grip Throttle
- Transmission Shift Control
- Service Brake Pedal
- Driver's Master Panel
  - Shutoff Switch
  - Tactical Idle Switch
  - Push-to start Switch
  - Started Light
- Driver's Instrument Panel
  - Engine Oil Temp High Light
  - Engine Oil Press Low Light
  - Engine Overspeed Light
- Fire Light
- First Shot Discharged Light
- RPM Gauge
- Gas Overtemp Light
- Second Shot Switch
- Transmission Oil Temp High Light
- Transmission Oil Press Low Light
- Vehicle Speed Gauge
- Fuel Gauge
- Low Fuel Level Light
- Tank Selector Switch
- Engine Oil Low Light
- Hydraulic Syst Malf Light
- Transmission Oil Low Light
- Engine Oil Clogged Filter Light
- Transmission Oil Clogged Filter Light
- Primary Fuel Clogged Filter Light
- Aircleaner Clogged Filter Light
- Rear Fuel Pump R Inoperative Light
- Rear Fuel Pump L Inoperative Light
- Fuel Control Faulty Light
- Remote Intercom Switches
- Engine Fire T Handle
- Crew Fire T Handle
- Unity Periscopes

Driver's Station - Minimum System

- Alert Panel
  - Master Warning Light
  - Reset Pushbutton
  - Master Caution Light
- Steer-Throttle Control
  - Steering Bar
  - Twist Grip Throttle
- Transmission Shift Control
- Service Brake Pedal
- Driver's Master Panel
  - Shutoff Switch
  - Tactical Idle Switch
  - Push-to Start Switch
  - Started Light
- Driver's Instrument Panel
  - Vehicle Speed Gauge
  - Fuel Gauge
  - Low Fuel Level Light
  - Tank Selector Switch
  - Remote Intercom Switches
  - Unity Periscopes

13 Appendix C. Interaction Device Controller (IDC)

Each position in the simulator (commander, gunner, driver, and loader) has a set of controls that may be simple switches, dials, throttle, brakes, etc. The simulation computer (SC) must have access to the settings of these controls and be able to set dials and displays.

One solution would be to have the SC have direct access to every control and display. This would require that the SC poll all controls periodically, and would also require wires from each switch be run to an interface board in the SC. This puts an additional burden on the SC, necessitates a large wiring harness in the simulator, and requires that each control and indicator be discretely wired.

To reduce the wiring harness in the simulator, relieve loading of the SC, and reduce production costs, a single chip microprocessor dedicated to each group of controls can be connected to the SC using a serial link. This same cable can supply power to the microprocessor and its associated circuitry. The result would be an eight-conductor cable running between the SC and each of the control groups, rather than discrete wiring for each switch running between the device and the SC. In addition, the SC now does not have to poll the various switch settings, but rather receives an interrupt as the change of state data arrives.

The desire is to have PC boards with most controls and the micro built as a single unit. In some cases, this would not be completely possible, since the controls of various groups are not all positioned in one confined location. Some discrete wiring will still be required, but will run only between the micro's PCB and the control.

13.1 Communication Protocol

Commands and data on the serial connections are 8-bit bytes. These bytes will be transmitted using RS232 protocol at 2400 baud. The basic operations to be performed are interrogating settings of indicators and switches and setting indicators. In normal operation, the control panel will report changes of state of any control (switch or A/D). At any time, the simulation system can request the state of a given control. This is necessary to determine the initial state of the control panels.

At present, three commands to the control panels are defined.

[0]	RESET	Restarts the uPC and reinitializes state.
[1]	SET	Sets the value of an indicator. Format is: 1,n,d where n is the indicator code and d is the value of that indicator.
[2]	GET	Retrieves the current setting of any switch or indicator. Format is: 2,n where n is the switch or indicator code.

All communication from the control panel are of the form n,d where n is the switch or indicator code and d is the current value. This is true for both changes of state and GET commands.

It may be desirable to add a command to set a hysteresis parameter for the controls with A/D inputs. This would be useful in reducing jitter in the controls. The microprocessor must also "debounce" all control switches to avoid sending a series of transitions to the SC.

One of the station IDCs should also handle requests for noises and communications failures. Certain codes on one of the IDCs will be set aside for the noise and communications subsystems. A SET command will cause a particular noise to be made or enable/disable the communications subsystem. The GET command will report the settings of these two pseudo-devices. A value of 1 when SETting the communications subsystem will enable it, 0 will disable it.

Noises vary in type and duration. Some noises are peculiar to an event, e.g., the cannon firing, while others are continuous background noises, e.g., the turbine running. The RESET command causes any noises to cease. Each noise command issued will last for an appropriate duration. Noises are initiated by issuing a SET command to an IDC. The code for the noise generator and the values corresponding to particular noises will be specified later by Perceptronics.

13.2 Implementation

Each major control panel will have a single chip microprocessor to deal with the various switches, indicators, analog-to-digital and digital-to-analog converters comprising the controls in the simulator. Each connection from the simulation computer to each control panel is a serial line (RS232 at 2400 baud) and power.

The connectors between the control and SC will be a DA15. The simulation side will be a DA15P and the control side will be a DA15S. The pin assignment is as follows (receive is the SC's input, the control computer's output):

DA15 Pin assignments

Pin #	Use
1	Ground
2	Receive
3	Transmit
4	+5 volts
5	+12 volts
6	-12 volts
7	+5 volts
8	Ground
9-15	Reserved

The +12 and -12 are required to produce proper RS232 levels. There may be additional voltages required by some of the displays or indicators in the control groups. If so, these must be added to the list of assigned pins. The present assignment uses eight wires with redundant +5 and ground to assure solid power and ground for the control systems.

13.3 Prototype Interaction Device Controller

While Perceptronics is constructing the actual IDC, it will be necessary to have a temporary IDC in use at BBN for software development purposes. To accommodate this requirement, a temporary IDC will be put together. This IDC will not have all (or possibly any) switches or controls, but will use a terminal to set the values of various switches and displays. A small microprocessor will be used to provide the RS232 lines to the SC and control the setting of various switches. In addition, the terminal will display the values of indicators such as the speedometer, fuel, tachometer, etc. This temporary IDC will provide sufficient functionality to proceed with software development, but will not have tank controls. The estimated time for development is ten (10) days.