Comput. & Graphics Vol. 14, No. 2, pp. 321–333, 1990
Printed in Great Britain.

0097-8493/90 $3.00 + .00
1990 Pergamon Press plc

Technical Section

A REAL-TIME, THREE-DIMENSIONAL MOVING PLATFORM VISUALIZATION TOOL

MICHAEL J. ZYDA, ROBERT B. MCGHEE, CORRINE M. MCCONKLE, ANDREW H. NELSON and RON S. ROSS
Naval Postgraduate School, Code 52, Dept. of Computer Science, Monterey, CA 93943

Abstract—Inexpensive, three-dimensional vehicle simulators are important visualization tools that can enhance training and serve as low-cost platforms for testing mobility expert system algorithms. The moving vehicle simulator is an interactive, real-time system that displays a dynamic, three-dimensional, out-the-window view of the terrain from any vehicle. The simulator has two modes of operation: stand-alone or networked. The networked mode facilitates a missile/target war gaming environment. The simulator can be easily adapted for use with a variety of computation resources on the network.

1. INTRODUCTION

Previous work in the Graphics and Video Laboratory of the Department of Computer Science at the Naval Postgraduate School included the production of a real-time simulator for the Fiber Optically Guided Missile (FOG-M)[1]. The FOG-M simulator displayed a real-time, three-dimensional, missile's eye view of terrain and vehicles driving over that terrain. The FOG-M simulator used digital terrain elevation data from the Defense Mapping Agency and a Silicon Graphics, Inc. IRIS-3120 graphics workstation.

The moving vehicle simulator (VEH) is a continuation of the FOG-M research[2]. The goal of the follow-on study is twofold. The first objective is to provide a stand-alone vehicle motion simulator. The second objective is to provide more realistic targets that, through networking, can be used by the FOG-M simulator. One noteworthy aspect of the simulator is that the operator can display the out-the-windshield view of any vehicle during program execution. The moving vehicle simulator has been incorporated into a Mobility Expert System (MES) and could easily be adapted for use by other simulators modeling off-road vehicle motion. It is the intent of this study to present the results of the design, development, and implementation of the moving vehicle simulator and the networking capabilities incorporated into the system.

2. BACKGROUND

The moving vehicle simulator models the motion of remotely piloted vehicles, such as jeeps, tanks, or trucks, one of which is designated the driven vehicle. The driven vehicle models a vehicle with an on-board video camera capable of transmitting live pictures of the battlefield to a distant operator's console. The moving vehicle simulator displays a real-time, three-dimensional, driver's view perspective of the terrain, and other vehicles. When networking is enabled, the FOG-M missile is also visible. An interactive user interface and a two-dimensional contour map display allow the operator to establish the desired simulator configuration (stand-alone or networked with the FOG-M simulator) and to define each vehicle to be used in the simulation. The vehicle locations, courses, speeds, and the selection of a driven vehicle are determined using a two-dimensional contour map display.

Once the simulation begins, a three-dimensional view of the terrain is displayed. The operator can interactively control the motion of the vehicle designated as the driven vehicle. The operator controls the driven vehicle's course, speed, and line-of-sight "look" direction by the knobs on a dial box. The viewing volume of the driven vehicle can be controlled by the mouse.

3. TERRAIN DATABASE

Both the moving vehicle simulator and the FOG-M simulator use a digital terrain elevation database provided by the Defense Mapping Agency (DMA) to draw the three-dimensional scene. This data is stored as an array of 16 bit data points that represent the terrain elevations of Fort Hunter-Liggett, California.

The 10 kilometer by 10 kilometer area of missile flight is sectioned into 100 meter squares, with each square consisting of two triangles (Fig. 1). The triangles are used to construct a colored, three-dimensional terrain display. Values for the triangles' coordinates are determined prior to missile flight.

4. GRAPHICS HARDWARE

The moving vehicle simulator is implemented using a Silicon Graphics, Inc. IRIS 3120 high-performance color graphics workstation. The workstation contains a Motorola 68020 microprocessor. The workstation also uses custom VLSI chips to provide hardware clipping and matrix transformations. The high-speed, pipeline architecture allows the performance of viewing, modeling, projection, and display device transformations at a much greater rate than would be possible in software. The graphics hardware can be conceptually depicted as three pipelined components: the applications/graphics processor, the geometry pipeline, and the raster subsystem. The geometry pipeline and the raster subsystem are controlled by the applications/graphics processor[3]. The IRIS provides a double buffer display system with a resolution of 1024 by 768 pixels.

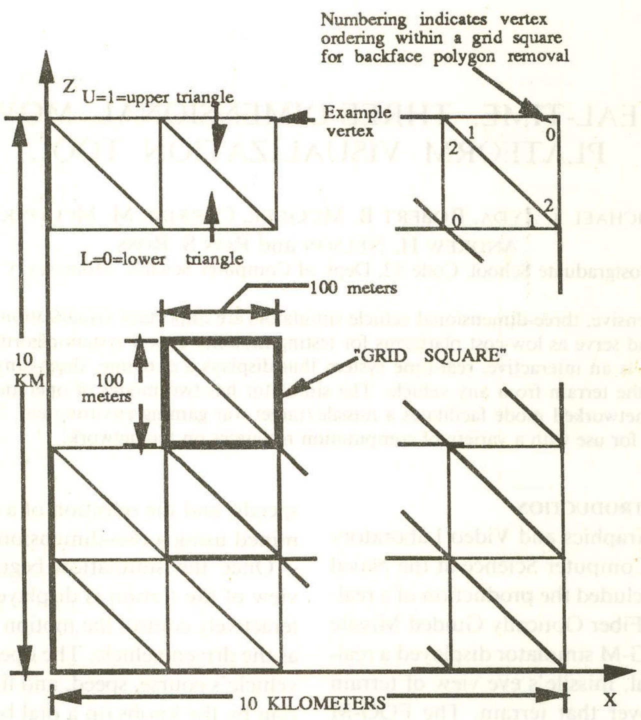


Fig. 1. Terrain polygons.

5. HIDDEN SURFACE ELIMINATION

Hidden surface elimination is accomplished by a real-time implementation of the painter's algorithm. The painter's algorithm simply draws objects in the scene in depth sorted (furthest to nearest) order [4, p. 266]. For the terrain, the correct polygon drawing order for hidden surface elimination is an easily computable function of the line-of-sight of the vehicle currently being operated (Fig. 2). Individual terrain grid squares are drawn as polygons based on the line-of-sight ordering. Vehicles located in the center of a grid square are drawn immediately after the grid square that they occupy is drawn. Vehicles crossing grid square boundaries are drawn only once. The grid square that they are drawn in is determined by using the line-of-sight ordering information. A vehicle is drawn in an adjacent grid square only if it is near certain edges. The edges are determined by the painter's algorithm. In Fig. 3, the line-of-sight from the driven vehicle A is as shown. With this line-of-sight, vehicles near a southern or eastern grid square edge are drawn after the adjacent grid square in that direction rather than in the grid

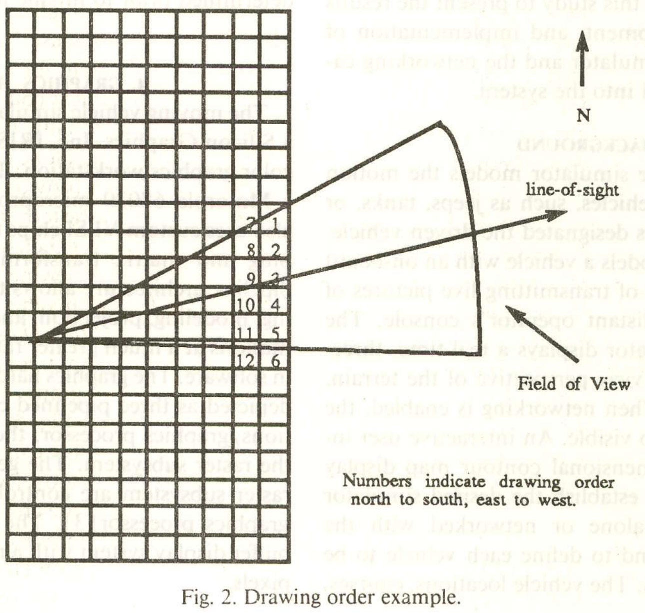
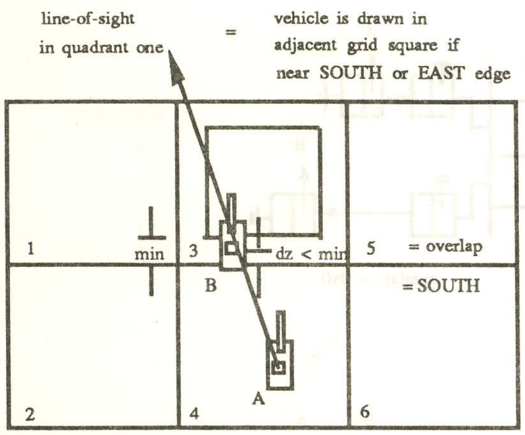

N
line-of-sight
7
1
8
2
9
3
10
4
11
5
12
6
Field of View
Numbers indicate drawing order north to south, east to west.

Fig. 2. Drawing order example.



Vehicle 'B' is near SOUTH edge =
draw it after grid square four

Fig. 3. Drawing in an adjacent grid square.

square the vehicles occupy. Vehicle B in Fig. 3 is located at the southern edge of grid square 3. Since the painter's algorithm draws grid square 3 before grid square 4, the part of the vehicle overlapping grid square 4 would be "painted over" by grid square 4 if the vehicle was drawn in grid square 3. To draw the vehicle correctly and both grid squares it overlaps, the vehicle must be drawn after grid square 4.

6. VEHICLES

In the moving vehicle simulator, the vehicles are created as graphical objects. Each polygon of each vehicle is drawn by defining its vertices and colors, and then drawing the polygon using a call to a polygon fill function. All objects are created using backface polygon removal and the painter's algorithm to display an undistorted view of a three-dimensional, light shaded object from any viewing angle above the ground plane.

Target vehicle objects (jeeps, trucks, tanks) are built during program initialization. After the objects are constructed, they are animated and oriented to the terrain. A vehicle's course and speed are used to calculate its new position based on the distance it would have traveled in the time required to refresh the screen. Each vehicle defined is associated with an element of one of three global two-dimensional arrays. There is one array for each of the three types of vehicles. The values stored in the arrays are the integer names of the graphical objects to be drawn in each terrain grid square. All vehicles present in one grid square are associated with the same element of the array. All commands required to draw each type of vehicle are collected into the same graphical object. Vehicles are displayed by drawing the terrain grid square and then accessing the appropriate two-dimensional array to draw the vehicles that are present in that grid square.

7. VEHICLE DATA STRUCTURES

The moving vehicle simulator uses two data structures to manage the vehicle display. A linked list of vehicle definition data is created before the display loop begins and is updated with each pass through the loop. Each structure in the linked list contains all the data required to transform and orient a vehicle object to the correct position on the terrain.

The second data structure manages vehicle hidden surface removal. A single two-dimensional array maintains the connection between the grid squares, and the order that the vehicles present in the grid square must be drawn. Each element in the array contains a list of pointers to records in the vehicle definition list for the vehicles that should be drawn immediately after drawing the terrain grid squares. The lists are maintained in depth sorted order (furthest to closest) from the driven vehicle. The grid square that a vehicle should be drawn in is determined by the vehicle's proximity to a grid square edge and the direction of the line-of-sight. As a result, a vehicle is drawn only once, regardless of its position on the terrain. As a vehicle overlaps a grid square, its position in the two-dimensional array changes. Fig. 4 shows how the array changes while maintaining the linked list depth sorted order. All the functions used to draw the vehicles and terrain are performed in the display loop. Each pass through the loop represents one frame of animation. By optimizing the functions, a frame rate that simulates a real-time display is achieved.

8. SOFTWARE IMPLEMENTATION

The moving vehicle simulator can be divided into two operational modes: stand-alone mode and networked mode. The stand-alone mode provides an environment where the operator can simulate driving vehicles over the selected terrain. In the networked mode, the moving vehicle simulator provides realistic targets for the FOG-M simulator.

8.1. *Stand-alone mode*

There are two fundamental sections of the stand-alone mode: the initialization phase and the vehicle driving simulation phase. The initialization phase provides an environment for vehicle definition and interactive input of vehicle course, speed, and position on the terrain. Additionally, the operator determines the desired mode (stand-alone or networking) in this phase. The driving phase provides as environment that dynamically updates the terrain displays in real time based on operator-controlled changes to the driven vehicle's speed, course, and viewing volume. The operator also designates the driven vehicle.

8.1.1. *Initialization phase.* The initialization phase is the interactive input component of the moving vehicle simulator program. The display screen is partitioned as shown in Fig. 5. The large area on the left part of the screen represents the two-dimensional contour map of the area over which the vehicles will operate. The contours are created from the elevation data in the DMA digital terrain elevation database. The map is color coded based on elevation points.

During this phase, the operator can define vehicles by moving the cursor on the contour map using the mouse. When the desired vehicle location on the map

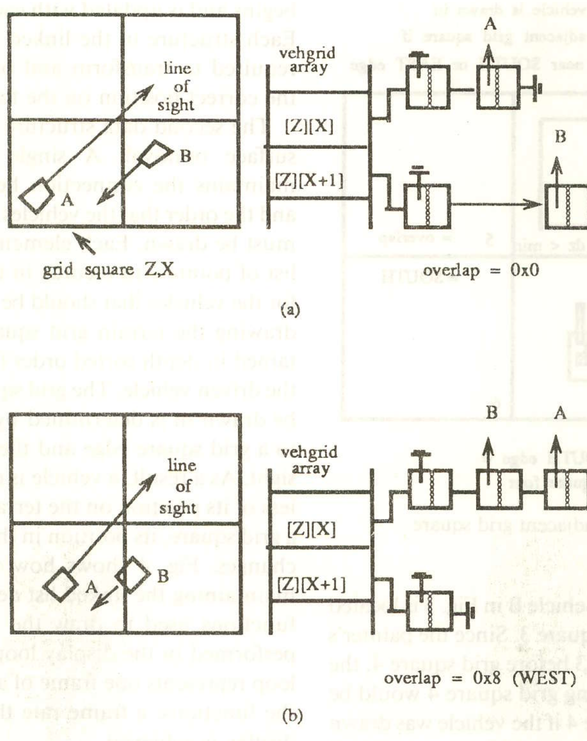


Fig. 4. Update vehicle grid.

is selected, the coordinates are locked in by pressing the right mouse button. An icon image of the vehicle appears on the map at the specified location.

8.1.2. *Vehicle driving simulation.* The driving simulation phase provides successive real-time terrain displays to the operator as the vehicle moves over the terrain. The simulation begins with the designation of a driven vehicle selected from the previously defined vehicles. The driven vehicle is selected by moving the cursor over the vehicle's icon image on the map and then depressing the right mouse button. Selection of a vehicle starts the display loop of the simulation. In networked mode, the vehicle simulator waits until the missile launch occurs before entering the display loop.

The driving display is partitioned as shown in Fig. 6. The large area to the left represents the out-the-win-

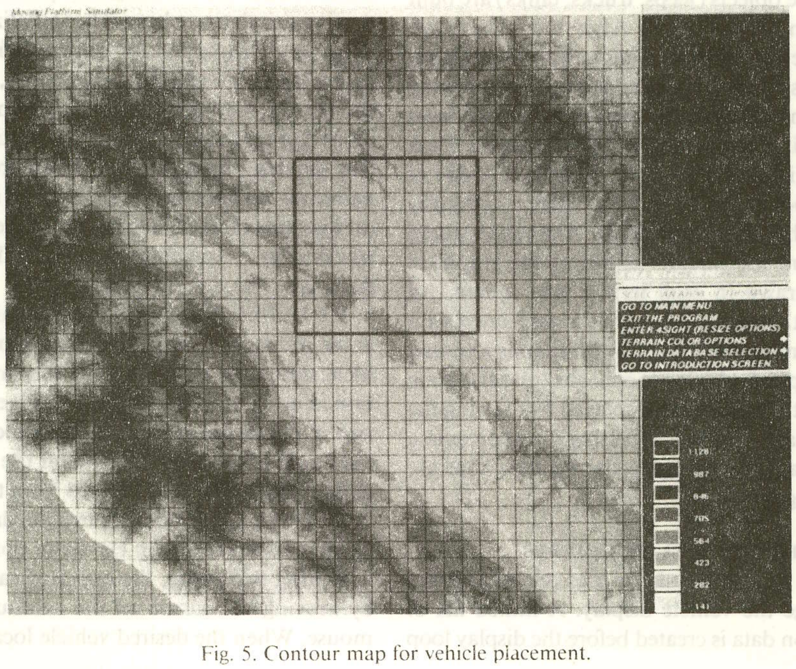

Moving Platform Simulator
GO TO MAIN MENU
EXIT THE PROGRAM
ENTER 4SIGHT (RESIZE OPTIONS)
TERRAIN COLOR OPTIONS
TERRAIN DATABASE SELECTION
GO TO INTRODUCTION SCREEN
1128
987
846
705
564
423
282
141

Fig. 5. Contour map for vehicle placement.

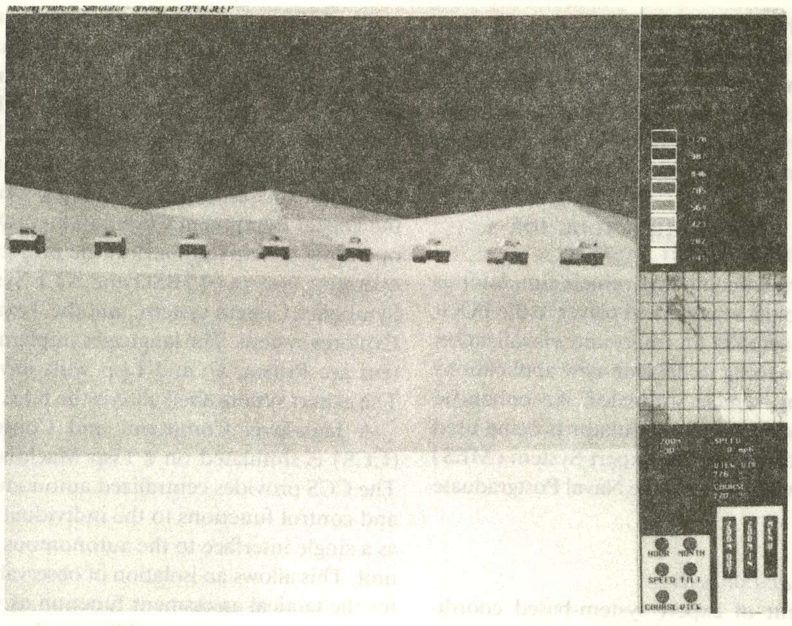


Fig. 6. Tanks in line formation.

dow view as seen from the driven vehicle. A popup menu is accessible that allows the operator to change vehicles or terminate the program. A contour map with the position of the driven vehicle and its viewing volume is displayed on the right, center section of the screen. The driven vehicle's speed, view direction, and available operator controls are shown in the lower right section of the screen.

8.2. *Networked mode*

The moving vehicle simulator is the first attempt at the Naval Postgraduate School to produce a network of real-time, interactive moving platform simulators. The network communication protocol selected is normal (blocking) socket I/O[5]. Blocking I/O allows synchronous operation of the FOG-M and moving vehicle simulators. A pair of sockets is used to transfer and guarantee delivery of the socket stream data between the two simulators. The moving vehicle simulator acts as the server to the client FOG-M simulator.

Operating the moving vehicle simulator in conjunction with the FOG-M simulator requires establishing network data paths. This is accomplished through the creation of dedicated sockets for read and write paths for both control and data. Failure to establish the communications paths causes the simulators to default to the stand-alone mode of operation.

Prior to missile launch, the missile operator's console is provided with relevant vehicle information, *i.e.*, the number and types of vehicles defined. Handshaking takes place after initial data transfer and before entering the display loop to allow either console to abort the simulation. If either simulation is aborted, the other can continue in stand-alone mode. After completion of the initial set-up, the FOG-M simulation console waits for the vehicle definition data from the moving vehicle simulator before allowing missile launch. The moving vehicle simulation waits for the launch event before entering the display loop to insure simulator synchronization. Regardless of the number of vehicles in the missile flight area, only the driven vehicle's information is sent to the missile console. The position of the other vehicles is predicted based on their initial position, course, and speed.

The missile simulator transfers a status flag to the moving vehicle simulator indicating if the missile is still in flight. If the missile is still flying, it sends missile position and course data. If it is no longer flying, it sends the identity of the vehicle destroyed.

9. SYSTEM FEATURES AND LIMITATIONS

Currently, the system allows only one console of each simulator type in a dedicated link arrangement to be networked together. To insure synchronization, a console cannot proceed past a socket read until the information is obtained. This lock-step execution prevents the vehicle console operator from changing the driven vehicle while the missile is in flight.

System performance for the networked mode, stand-alone mode, and the mobility expert system (MES) is shown in Table 1. *Static* refers to the type of vehicle

Table 1. Display update rates.

Simulator mode	Number of vehicles	Frames per second
Networked	1 (static)	2.6
	10 (static)	1.9
	1 (dynamic)	1.4
	10 (dynamic)	1.2
Stand-Alone	1 (static)	5.7
	10 (static)	4.0
	1 (dynamic)	5.3
	10 (dynamic)	4.3
MES	1 (dynamic)	3.7
	10 (dynamic)	3.3

objects drawn in the original FOG-M simulator. *Dynamic* refers to vehicle objects that more closely reflect normal vehicle dynamics over natural terrain. The vehicle dynamics modeled in the MES are more complicated than the dynamics modeled in the other simulators, resulting in a slower frame update rate.

10. VEH AS A VISUALIZATION TOOL FOR A MOBILITY EXPERT SYSTEM

Above, we describe the moving vehicle simulator as either stand-alone or as a networked player to the FOG-M simulator. It is actually an important visualization tool. Research is ongoing to develop new applications around the moving vehicle simulator. An enhanced version of the moving vehicle simulator is being used in conjunction with a Mobility Expert System (MES) currently under development at the Naval Postgraduate School.

10.1. *MES goals and objectives*

The development of expert system-based coordination algorithms for groups of autonomous vehicles is the major objective of the MES project. The second objective is to develop the software necessary to create motion simulation of the system using realistic vehicle dynamics over a computer generated terrain model. For purposes of this study, the prototype system developed closely follows the model of the FMC autopilot[6]. The program hierarchy is shown in Fig. 7.

The MES is a system using four different computer architectures, three programming languages, four networking packages, three operating systems, and an expert system shell. The four computer architectures used are: the Symbolics 3600 line of Lisp Machines, the Texas Instrument Explorer Lisp Machine, the Silicon Graphics, Inc. IRIS-3120 Graphics Workstation, and the Digital Equipment Corporation VAX 11/785. The operating systems utilized in the project are the Unix operating system (4.3BSD and ATT System V.3), the Symbolics Genera system, and the Texas Instruments Explorer system. The languages implementing the system are Prolog, C, and Lisp with flavor extensions. The expert system shell used is the KEE expert system.

A high-level Command and Control Subsystem (CCS) is simulated on a Lisp Machine and a VAX. The CCS provides centralized autonomous command and control functions to the individual tanks and acts as a single interface to the autonomous vehicles in the unit. This allows an isolation of observable phenomena for the tactical assessment function as well as centralizing the focus of one problem in the research area.

Simulated tanks with the characteristics of the existing FMC Autonomous Land Vehicle are modeled as in [6]. The model is conceptually organized into two distinct parts: (1) the graphics instantiation, with vehicle controller functions on the IRIS, and (2) the rule-based, expert system behavior, implemented on the Lisp Machines. The tanks operate autonomously

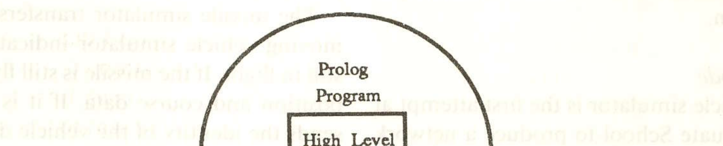
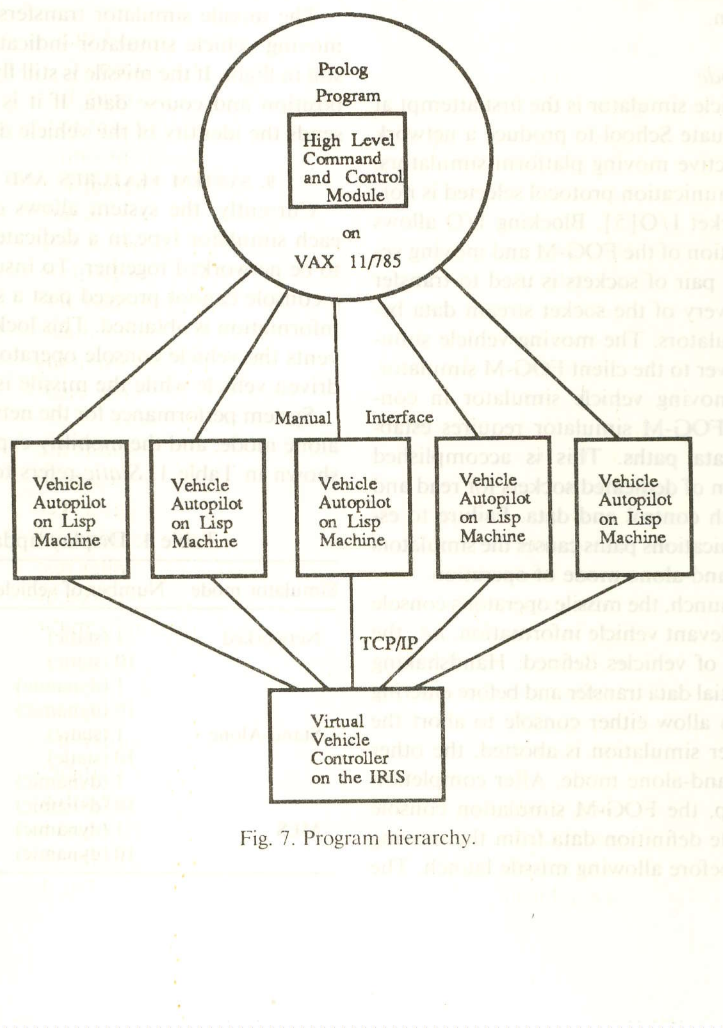


Prolog Program
High Level Command and Control Module
on VAX 11/785
Manual Interface
Vehicle Autopilot on Lisp Machine
Vehicle Autopilot on Lisp Machine
Vehicle Autopilot on Lisp Machine
Vehicle Autopilot on Lisp Machine
Vehicle Autopilot on Lisp Machine
TCP/IP
Virtual Vehicle Controller on the IRIS

Fig. 7. Program hierarchy.

```
Loop

  Check for commands from the Command and Control Subsystem.
    If change in formation, acquire rules and facts necessary from disk storage and implement.

  Perform a visual scan of the environment.

  For each object identified:

    Establish its position in reference to the tank's body coordinate system.

    Approximate its future location at beginning of next iteration of the algorithm.

    Produce low level observations about the object as input to the task generator.

  EndFor

  Generate tasks in the task generator using the low level observations and knowledge and rules necessary
    to complete currently assigned goals.

  Display diagnostic information and explanations for each task generated.

  Execute communications tasks to Command and Control subsystem.

  Execute tasks generated by communicating sequences of vehicle steer and reference velocity commands
    to the vehicle controller residing on the IRIS.

EndLoop.
```

Fig. 8. Autonomous tank control algorithm.

in much the same way as the FMC vehicle[6]. Specifically, each tank possesses a simulated vision capability, an autopilot, and the ability to send vehicle steering and reference velocity commands to a vehicle controller.

10.2. *Autonomous tank rules*

Individual tanks perform according to the algorithm presented in Fig. 8. The autopilot possesses capabilities in addition to those being developed at FMC[7]. These extra capabilities allow the vehicle to act as an integral part of a tactical autonomous unit. A tank's designated place in a tactical formation is based on the commands sent to it from the lead tank. The tank maintains its station in the formation until it receives new commands. Currently the tanks use three sets of simple rules that allow the vehicles to assume a line, column, or file formation[8]. For each formation, each tank possesses knowledge about who it is, the type of formation, its guide vehicle, and the vehicles that should be to its flanks, front, and rear. Rules for each formation are divided into four functional categories: collision avoidance, speed determination, direction determination, and stationing. These rules are presented in Figs. 9 through 12.

An autonomous tank is comprised of a set of functions that reside on a Lisp machine. The autonomous tank's controller and graphics object reside on the IRIS. Each Lisp machine controls a graphically rendered tank on the IRIS battlefield during a simulation run. The Lisp functions perform the algorithms presented in Figs. 8 through 12. Each Lisp machine generates task commands that are sent to the individual tank that it controls. The Lisp machines also determine the approximate time interval required for the tank to respond to the task command.

The tanks perform a simulated visual scan of the environment in the IRIS and produce high-level observations about the battlefield. These observations are used to perform tactical assessments and create tasks to accomplish goals using rule-based inference engines. A rule-based inference engine is a program that processes if<circumstances>then<do-task> type expressions. These expressions are constructed through the interrogation of an expert. Typical tasks, such as those generated for formation keeping, are vehicle referent velocities and directions. These tasks are transmitted to the vehicle controller residing on the IRIS. The vehicle controller then executes the tasks and communicates feedback information to the requesting Lisp Machine.

10.3. *A single iteration of start-the-battle*

Fig. 13 presents a single iteration of the tank algorithm for tank 1 operating in conjunction with two other vehicles, tank 2 and tank 3. The information in Fig. 13 is taken from the display of the Lisp Machine designated as tank 1. Fig. 13a shows both tank 1 and

```
Avoid Collision To The Right:
  If
    the vehicle is or will be too close to an object, and the object is to the right of the vehicle,
  Then
    move to the left.

Avoid Collision To The Left:
  If
    the vehicle is or will be too close to an object, and the object is to the left of the vehicle,
  Then
    move to the right.

Avoid Collision Ahead:
  If
    the vehicle is or will be too close to an object, and the object is ahead of the vehicle,
  Then
    If
      not enough time to maneuver,
    Then
      Stop.
    ElseIf
      able to maneuver,
    Then
      maneuver around object in flank with greatest maneuvering room.

Avoid Collision From Behind:
  If
    the vehicle is or will be too close to an object, and the object is behind the vehicle and closing,
  Then
    match the object's speed.
```

Fig. 9. Collision avoidance rules.

tank 2's grid coordinates, course, speed, and information about tank 2's position, course, and speed relative to tank 1. Fig. 13b shows the same information for tank 3. Fig. 13c shows the rules needed to move a tank to the right. Fig. 13d shows the rules used for a line formation for maintaining a separation interval between two tanks. Fig. 13e shows the rules used when tank 2 is the guide tank and tank 1 is too far ahead of the guide tank. As a result, tank 1 is ordered to stop. Once the guide tank catches up, another set of rules (not shown) is used to order tank 1 to increase speed.

The tanks reason about the IRIS battlefield world

```
Change Speed:
  If
    vehicle is on course with its guide vehicle, and vehicle is behind or ahead of its station,
  Then
    change speed to move vehicle to position by next iteration of tank algorithm.

Match Speed:
  If
    vehicle is on course with its guide vehicle, and vehicle is on station with its guide vehicle,
  Then
    match speed of the guide vehicle.

Stop:
  If
    guide vehicle is stopped, and vehicle on station with guide vehicle,
  Then
    stop vehicle on station.
```

Fig. 10. Speed determination rules.

```
Turn Left:
  If
    vehicle is off course from its guide vehicle and relative right to the direction of guide vehicle's course,
  Then
    turn left the angular difference to come about.

Turn Right:
  If
    vehicle is off course from its guide vehicle and relative left to the direction of guide vehicle's course,
  Then
    turn right the angular difference to come about.
```

Fig. 11. Direction determination rules.

relative to their own individual body coordinate systems. The tanks reason about time by approximating positions, dispositions, and possible intentions of objects in view during possible future event time frames. Tanks also continuously reevalute their individual circumstances as well as their vehicle controller's response time to a direction or velocity command. This allows a tank to predict and address future events. Fig. 14 provides an example.

In Fig. 14, we show the coordinates of tank 3 relative to tank 1 at time t. Also shown are the coordinates of tank 3 relative to tank 1's predicted future location at time t′. Tank 1 will be too close to tank 3 because the horizontal interval distance will exceed the value of a constant measure, called proper interval, as tank 1 approaches tank 3 from behind. The proper interval is the required distance between two tanks in the formation. This distance varies depending on the type of formation being executed. When the distance between the two tanks is less than the proper interval, a task is generated by tank 1 to increase the distance between the two tanks.

10.4. *A typical test mission*

Figs. 15 through 18 illustrate a typical test mission. Fig. 15 depicts the movement from an assembly area. The initialization phase for the IRIS has been conducted, the tactical assessment carried out, and the Lisp machines have been initialized to drive all but one of the tanks in the unit. The guide vehicle for the unit, driven by a human operator on the IRIS, has been given an initial direction and speed. The jeep was then selected to view the formation as it turned to its left to assume a column formation. The picture was taken from the jeep.

Fig. 16 depicts the column after crossing the line of departure and conducting movement to contact (going out and engaging the enemy). The guide vehicle is the lead tank in the column. To obtain the picture, the jeep was driven to a known destination of the lead tank. The jeep then was positioned to get a view as the column approached.

Fig. 17 depicts the actions at the final coordination line. The unit deployed into a line formation and is about to move through the objective. This deployment

```
Close Right With Guide:
  If
    vehicle is too far from guide, and vehicle is left of guide, and guide vehicle is normally vehicle's right vehicle,
  Then
    move to the right.

Close Left With Guide
  If
    vehicle is too far from guide, and vehicle is right of guide, and guide vehicle is normally vehicle's left vehicle,
  Then
    move to the left.

Assume Correct Position in Relation to Guide:
  If
    vehicle is on course with guide, and vehicle is left/right of guide, but vehicle should be right/left of guide,
  Then
    drop behind guide,
    turn 90 degrees right/left,
    proceed until past guide,
    turn 90 degrees left/right.
```

Fig. 12. Stationing rules.

```
> (gettanks "lineformation")
T
> (start-the-battle 1 3)

Tank #1 now conscious.
Tank #1's location = (5300, 1743).
Tank #1's speed = 0.0
Tank #1's course = 302 degrees.
Tank #1's course relative to compass north = -58 degrees.
Comparison tank is tank #2.
Tank #2's location = (5409, 1721).
Tank #2's speed = 1.0
Tank #2's course = 303 degrees.
Tank #2's course relative to tank #1's course = 1 degree.
Tank #2's transformed position relative to tank #1 = (39, -104).
Predicted relative transformed position of tank #2 when tank #1 again becomes conscious = (39, -84).
Relative distance between tank #1 and tank #2 when tank #1 again becomes conscious = 19.
(13a)

Tank #1's location = (5300, 1743).
Tank #1's speed = 0.0
Tank #1's course = 302 degrees.
Tank #1's course relative to compass north = -58 degrees.
Comparison tank is tank #3.
Tank #3's location = (5432, 1809).
Tank #3's speed = 1.0
Tank #3's course = 302 degrees.
Tank #3's course relative to tank #1's course = 0 degrees.
Tank #3's transformed position relative to tank #1 = (126, -76).
Predicted relative transformed position of tank #3 when tank #1 again becomes conscious = (126, -57).
Relative distance between tank #1 and tank #3 when tank #1 again becomes conscious = 19.
(13b)

(RULE CLOSE-RIGHT SAYS TASK MOVE-TO-RIGHT 1)
(RULE STOP SAYS TASK STOP 1)
(13c)

(TASK MOVE-TO-RIGHT 1 BECAUSE)                    (TASK STOP 1 BECAUSE)
(RIGHT VEHICLE IS 2)                              (1 WILL BE AHEAD OF 2)
(1 IS LEFT OF 2)                                  (GUIDE VEHICLE IS 2)
(1 WILL BE LEFT OF 2)                             (VEHICLE IS 1)
(1 WILL BE TOO FAR FROM 2)                        (FORMATION IS LINE)
(GUIDE VEHICLE IS 2)                              (13e)
(VEHICLE IS 1)
(FORMATION IS LINE)
(13d)
```

Fig. 13. Single iteration of Start-the-battle.

was effected with the help of manual intervention. The guide tank was stopped at the final coordination line by a human operator. This forced the column to halt by initiating certain station keeping rules. The function application of Start-the-battle was allowed to expire upon each Lisp machine. A new formation was then acquired by each Lisp machine. The function Start-the-battle was then reapplied upon each Lisp machine. The human operator assumed control of the guide vehicle while the autonomous, Lisp machine-driven tanks then assumed their positions in the line formation after about 30 seconds of maneuvering.

Fig. 18 depicts the line of tanks as they assault an objective. The line is sweeping past the stationary jeep from which the picture was taken.

10.5. *MES implementation*

The MES system is distributed across the various specialized architectures in accordance with hardware capabilities. Thus, it was possible to create an entirely satisfactory real-time system at low cost. The current suite of equipment allows up to five individual tanks to operate on the battlefield represented on the IRIS.

Performance bottlenecks occur during communication processing on the IRIS. This is because each tank spawns a send and receive process to communi-

```
Tank #1 now conscious
Tank #1's location = (4474, 2412).
Tank #1's speed = 1.46
Tank #1's course = 302 degrees.
Tank #1's course relative to compass north = -58 degrees.
Comparison tank is tank #3.
Tank #3's location = (4453, 2428).
Tank #3's speed = 1.0
Tank #3's course = 302 degrees.
Tank #3's course relative to tank #1's course = 0 degrees.
Tank #3's transformed position relative to tank #1 = (3, 27).
Predicted relative transformed position of tank #3 when tank #1 again becomes conscious = (3, 7).
Relative distance between tank #1 and tank #3 when tank #1 again becomes conscious = 19.
(14a)

(RULE AVOID-COLLISION-TO-RIGHT SAYS TASK MOVE-TO-LEFT 1)
(14b)


(TASK MOVE-TO-LEFT 1 BECAUSE)
(1 WILL BE LEFT OF 3)
(1 WILL BE TOO CLOSE TO 3)
(VEHICLE IS 1)
(FORMATION IS LINE)
(14c)
```

Fig. 14. Reasoning about future events.

cate to a Lisp machine. The performance bottlenecks on the Lisp machine side are in relation to the sequential nature of the command and control system's execution. The problem is that the vision and inference operations are not concurrent or continuous. The Lisp machine must ask the IRIS for vision information and then wait until the IRIS collects and returns the vision information. Once it has the information, it uses the information to make inferences about the tank it is controlling relative to the other tanks on the battlefield.

11. CONCLUSION

We have described how one extends the capabilities of inexpensive three-dimensional visual simulators on individual workstations to the networked workstation environment. Individual graphics workstations are

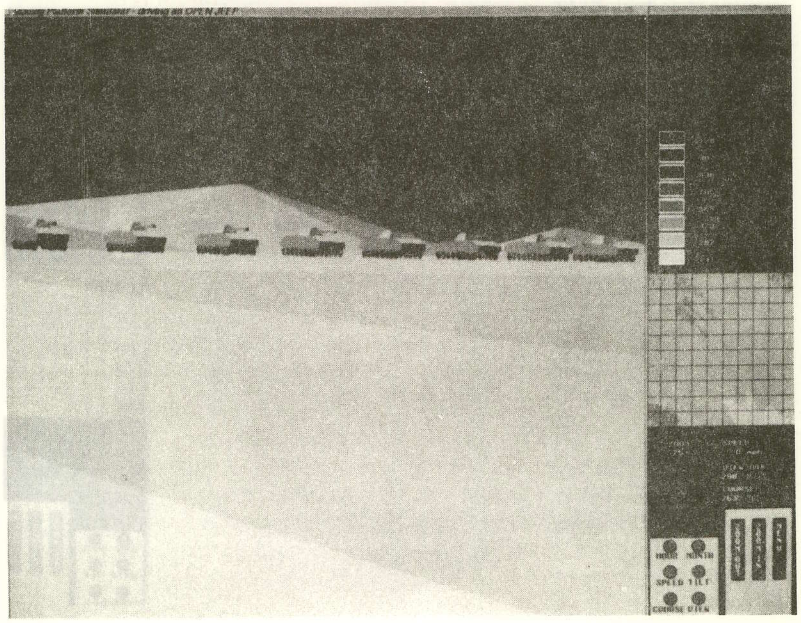


Fig. 15. Moving to the line of departure.

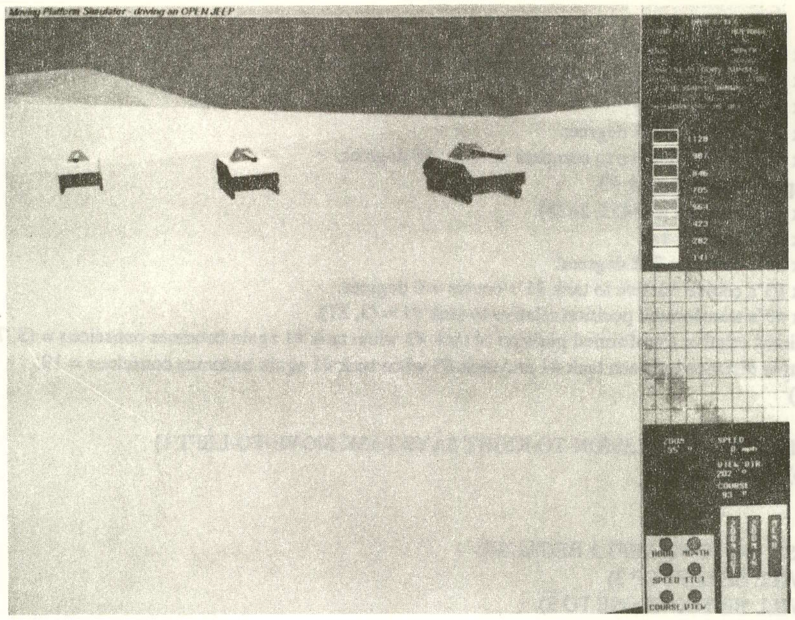


Fig. 16. Crossing the line of departure.

easily grown out of as our applications become more sophisticated. We grow out onto a network of workstations to allow for other players or to partition our system into processes computed by separate machines. For the system described above, the partitioning across machine boundaries has been expensive due to the lack of readily available networked graphics and computing software facilities. We expect this to change as high-performance graphics workstation manufacturers recognize the importance of distributed graphics and computational functionality. Ideas such as location-independent computing and location-independent graphical objects are a step in the right direction.

We have also shown how the notion of inexpensive three-dimensional visual simulators as visualization tools can lead to better understanding for typically graphics-less areas such as expert systems. Three-dimensional simulators that can be readily "plugged-into" diverse computational environments present a viable alternative for the future. To accomplish this goal, we need to make our three-dimensional visual simulators inexpensive and adaptable.

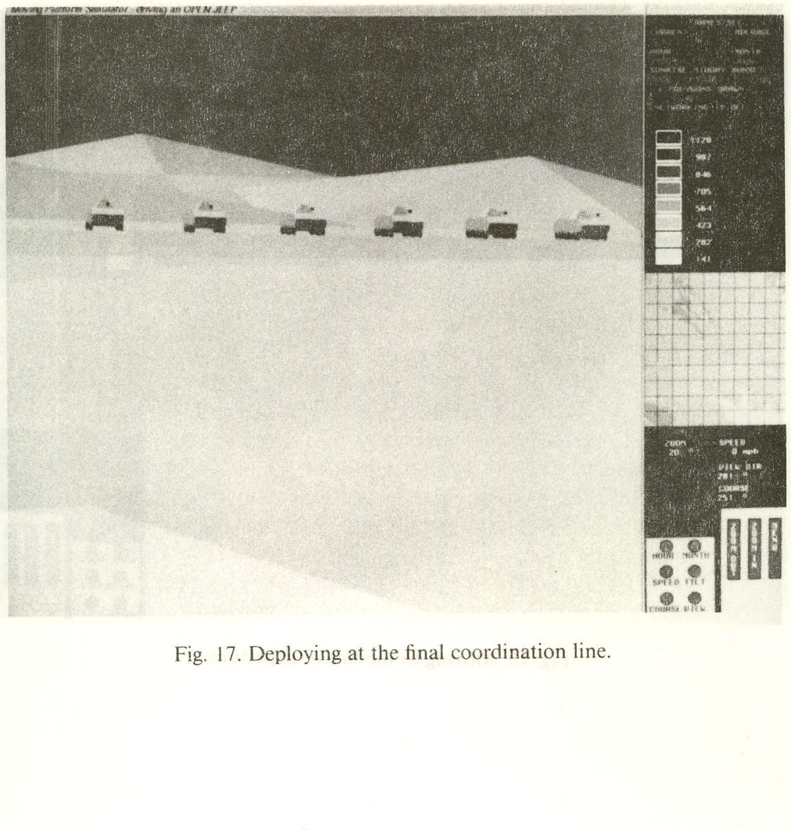


Fig. 17. Deploying at the final coordination line.

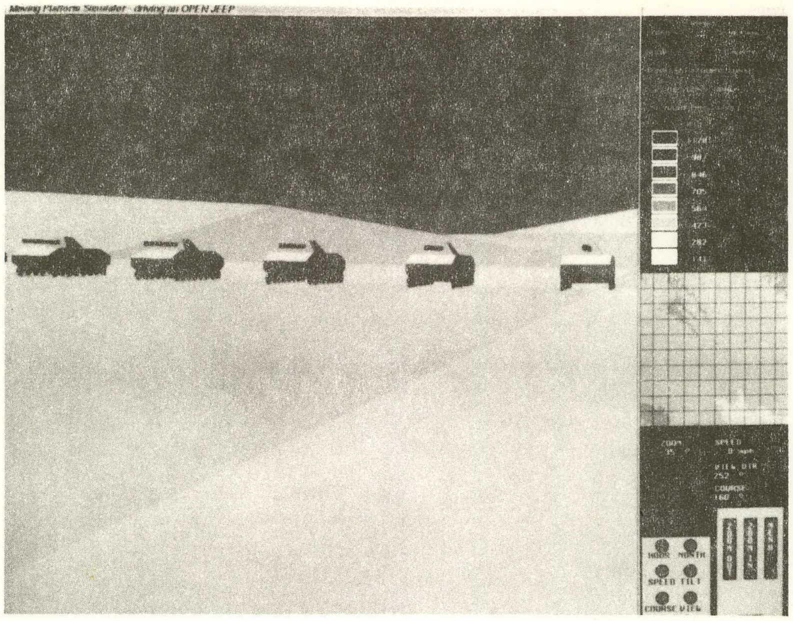


Fig. 18. Assaulting the objective.

Acknowledgements—We wish to acknowledge the students involved in this effort, especially Michael R. Oliver, David J. Stahl, Jr., Dale G. Streyle, and Douglas B. Smith.

This work was supported by the U.S. Army Engineer Topographic Laboratories, Fort Belvoir, the U.S. Army Test and Experimentation Command, Fort Ord, the Naval Ocean Systems Center, San Diego, and the Naval Postgraduate School's Direct Funding Program.

REFERENCES

1. M. J. Zyda, R. B. McGhee, R. S. Ross, D. B. Smith, and D. G. Streyle, Flight simulators for under $100,000. *IEEE Computer Graphics and Applications* **8**(1), 19–27 (January 1988).
2. M. R. Oliver and D. J. Stahl, Jr., *Interactive, Networked, Moving Platform Simulators.* M.S. Thesis, Naval Postgraduate School, Monterey, CA, December 1987.
3. Silicon Graphics System Documentation, *Product Specifications,* Silicon Graphics Incorporated, Mountain View, CA (1985).
4. M. Baker and D. Hearn, *Computer Graphics,* Prentice Hall, Englewood Cliffs, NJ (1986).
5. T. Barrow, *Distributed Computer Communications in Support of Real-Time Visual Simulations.* M.S. Thesis, U.S. Naval Postgraduate School, Monterey, CA, June 1988.
6. J. Nitao and A. Parodi, A real-time reflexive pilot for an autonomous land vehicle. *IEEE Control Systems Magazine* **6**(1), 14–23 (February 1986).
7. J. Mitchell, An autonomous vehicle navigation algorithm. Proc. SPIE, *Applications of Artificial Intelligence,* **485,** 153–158 (1984).
8. A. H. Nelson, and C. M. McConkle, *A Prototype Simulation System for Combat Vehicle Coordination and Motion Visualization.* M.S. Thesis, Naval Postgraduate School, Monterey, CA, June 1988.