# NPSNET: JANUS-3D Providing Three-Dimensional Displays for a Two-Dimensional Combat Model

David R. Pratt, Jon C. Walter, Patrick T. Warren, and Michael J. Zyda
Graphics and Video Laboratory
Naval Postgraduate School, Department of Computer Science
Monterey, CA 93943-5100 USA

## Abstract

*This paper discusses the integration of the Army's existing combat modeling tool, JANUS(A), with the real-time three-dimensional graphics display offered by NPSNET. A scripting tool capable of rendering JANUS(A) scenarios previously executed in the traditional two-dimensional model is discussed. This replay capability allows the gamer/analyst the ability to watch the three-dimensional battle unfold from any position on the battlefield. Also, the implementation of a real-time, networked link from the two-dimensional JANUS model to NPSNET is detailed. This link involves an Ethernet connection from a Sun workstation, which houses the two-dimensional model, to a Silicon Graphics workstation used for rendering the real-time three-dimensional simulation..*

## Introduction

In 1991, the U. S. Army Training and Doctrine Command selected JANUS(A) as the simulation software standard for training at company and platoon levels, using the battalion commander as the senior trainer. It is also used in a seminar role by the Command and General Staff College to train new battalion and brigade commanders on the principles of synchronized combined arms operations [JANU 91]. JANUS(A) is fielded throughout the world and, because of its ability to accurately model complex combat scenarios, is widely used by trainers and analysts in numerous applications which include combat training, studies of combat operations, combat development, testing of new equipment, and research and development.

Despite its great success and huge popularity as a combat development and testing tool, JANUS(A) analysts and developers realized the need for a three-dimensional view of the battlefield to validate their results. One project, where the Army finds the three-dimensional view useful, is the new system testing of the M1A2 main battle tank at Fort Hunter Liggett, California. As part of this testing, the U. S. Army TRADOC Analysis Command, Monterey (TRAC-MTRY), used JANUS(A) as a tool to model the operational testing of the M1A2 tank, before it was actually tested on the ground [PAUL 92]. A three-dimensional capability would help validate this data by verifying events, such as tank positions, direct fire engagements, and line of sights between weapon systems.

In addition to its obvious usefulness with the development of new combat systems, there are many other applications for this system. Because of past virtual reality successes, such as SIMNET, users knew that a three-dimensional view, in support of JANUS(A) exercises, would greatly enhance training simulations. Almost as good as being on the actual terrain, a three-dimensional world would give the gamer/analyst the ability to watch the battle unfold from any position on the battlefield. In addition to standard scenarios, this ability could be used, in conjunction with JANUS(A), for three-dimensional 'after-action reviews' at Combat Training Centers, such as the National Training Center, Fort Irwin, California. Likewise, previously fought battles, such as the 'Battle of 73 Easting' of Desert Storm fame, could be reenacted on JANUS(A), and refought in the three-dimensional simulation, allowing for several 'what-if' conditions.

## JANUS(A) Description

JANUS(A), is an interactive, computer based, war-gaming simulation of combat operations conducted at the brigade and lower level in the United States Army [JANU 86]. JANUS(A) is a "two-sided, interactive, closed, stochastic, ground combat simulation" [JANU 91]. It is termed 'two-sided' because it allows the simulation of two opposing forces. These two forces, the Blue force and the Red force, are simultaneously directed and controlled on separate monitors by two different sets of players. Each monitor displays only the vehicles pertaining to its side, plus the opposing vehicles which are directly observed by its vehicles. Therefore, the model is classified 'closed' because the friendly force player does not know the complete disposition of the opposing forces. The model is 'interactive' because each player monitors, directs, reacts to, and redirects all key actions of the simulated units under his control. Once a scenario is started, certain events in the game, such as direct fires and artillery impacts, are 'stochastically' modelled, which means that they act according to the laws of probability, and thus are different for every scenario run. The principal modeling focus in JANUS(A) is on military systems that participate in maneuver and artillery operations on land, thus the term 'ground combat simulation'.

JANUS(A) is composed entirely of Army-developed algorithms and data to model combat processes. The multitude of programs which belong to JANUS(A) consist of approximately 200,000 lines of code written entirely in VAX-11 FORTRAN, a structured Digital Equipment Corporation (DEC) extension of ANSI standard FORTRAN-77 [JANU 91].

## NPSNET-JANUS Integration

The Naval Postgraduate School's Computer Science Department created a system to render three-dimensional, real-time simulations of military vehicles and terrain databases known as NPS Networked Vehicle Simulator (NPSNET) [ZYDA 91] [ZYDA 91a] [ZYDA 92] [FALB 92]. Specific attributes of the system include creating a three-dimensional, real-time, virtual world at a cost of well under $100,000 [ZYDA 92]. Furthermore, it possesses the ability for quick adaptation with applications involving other modeling systems.

NPSNET currently provides three modes of combat modeling. First, the model can be played in a networked, multi-user mode. In this mode, one or more players can chose and maneuver any vehicle in the three-dimensional virtual world via a 'Space Ball'. They can decide to fight with or against one another over the network. Additionally, a player can choose to fight a set of semi-automated forces. Players can view the world from the perspective of the vehicle commander or from an observer controller vehicle. The second modeling mode NPSNET uses is a script. With this method, the user can prepare scripted scenarios of combat operations and view the results in three-dimensions. The last mode includes receiving inputs over an Ethernet network from other three-dimensional combat models such as SIMNET. In this mode, NPSNET can fully interact with the other combat model.

NPSNET provides an assortment of U.S. and foreign weapons system models. Currently, vehicle models range from U.S. M1A1 tanks and A-10 jet aircraft to Soviet made T72 tanks and BMPs. All of these vehicles are formatted and rendered using the NPSOFF system [ZYDA 91a].

NPSNET uses the JANUS(A) terrain database to construct the three-dimensional terrain. The JANUS(A) terrain database is merely a collection of grid coordinates with an elevation assigned to each coordinate. In order to store these elevations in a sequential file format, JANUS(A) divides the terrain into a grid. The spacing between the horizontal and vertical lines is known as the resolution value. Currently, JANUS(A) can support resolution values of 12.5, 25, 50, 100, and 200. Each intersection is labeled by the cartesian coordinates of the horizontal and vertical line that make up the intersection. These grid intersections are known as checkpoints. The ground elevation is recorded for each grid cell, and, if a tree or city is present in a certain grid, this is also recorded along with its respective height factor. JANUS(A) references the cartesian coordinates for each checkpoint using its respective UTM coordinate.

NPSNET utilizes two techniques to render three-dimensional terrain, using either triangular-mesh or discrete polygons as basic building blocks. Mesh terrain is created using the built in "IRIS" function for drawing a triangular mesh, commonly referred to as "t-mesh". Each checkpoint is treated as one of the three points of a triangle. The t-mesh terrain skin is built by moving from the left to the right of the map, then from the bottom to the top of the map. The triangular-mesh terrain provides an efficient and simple terrain skin for the virtual world, but currently does not allow roads and rivers to be drawn without significant tearing.

In order to realistically provide the ability to render roads and rivers, NPSNET instead utilizes the discrete polygon (*polygonized*) terrain model [FALB 92]. Because each polygon is a discrete element, each one can be manipulated -- or any object on it -- with great precision.

A grid node is the basic building block for polygonized terrain. Each node is made up of two triangles, an upper and a lower. The numbering for each grid node uses the upper-left hand corner of the square as the local origin, with each corner of the node representing a checkpoint from the database. Each side of the *basic* grid node is equal to the resolution value. This three-dimensional view of the terrain is a fairly good representation, with the accuracy increasing with the lower resolution values (smaller grid spacing).

All objects (such as trees, vehicles and buildings) are rendered using a system known as NPSOFF [ZYDA 91a]. These stationary objects and polygonized terrain data are stored in files indexed by coordinates which match their corresponding terrain node. Moving objects are stored in an array assigned to each terrain node. Their positions are updated each time the graphics loop is executed [MACK 91].

## Generation of a Three-Dimensional Script

In the traditional JANUS(A) model, vehicles are represented on a two-dimensional screen using a simple icon. Depending on whether a vehicle is classified as "friend" or "foe", this icon is permanently faced to the left or right and no indication is given to its direction of travel, its orientation, or its gun tube/sight orientation [JANU 86].

All significant events which occur during a JANUS(A) scenario are recorded in five disk files (movement, direct fires, artillery fires, kills, detections), known as post-processor files. These post-processor files are converted to create a realistic and accurate script for the three dimensional simulation, NPSNET. The major hurdle to overcome for this conversion, is that a two-dimensional game does not require (and in this case, does not produce) all of the information needed to define what a vehicle is actually doing at every moment in the game.

In general these files list the *x* and *y* coordinates of a vehicle at the time an event occurs. Additionally, who the vehicle shot, the location of the target, the location of impact for each artillery shot, and the clock time for each of these events is included with these listings [JANU 86]. When the actual JANUS model is running, it is possible, upon request, to find out the speed of a vehicle and its line-of-sight fan. However, this information is not recorded in the post-processor files.

In order to recreate a viable three-dimensional representation of the game, new information about what each vehicle was doing between recorded events must be inferred. The *C Language Integrated Production System* (CLIPS) expert system shell was chosen as the inference engine to generate this new information [GIAR 89]. CLIPS was chosen because of the considerable amount of pattern matching necessary.

The first step in the information inference process is to determine the data that NPSNET requires for rendering. One of the capabilities of NPSNET is to display and move vehicles according to a script. To read the script, NPSNET first compares the current game clock with the time assigned to the top element in the script file. If the game clock is greater than or equal to the event time, then the event is read and its instructions are implemented. To simplify the processing of a script, each line in the script file is considered an event. The information contained in each event line is as follows:

- Time
- Vehicle Number
- Vehicle Type
- X coordinate
- Z coordinate (referred to as the Y coordinate in a 2D world)
- Vehicle orientation (in positive radians)
- Weapon orientation (in positive radians)
- Speed (meters per second)
- Shot fired (1= Yes; 0 = No)
- Alive (1= alive; 0 = killed)
- Elevation (ground vehicle = 0; helicopter = 100 meters; plane = 200 meters)

After NPSNET reads an event, it assigns the event speed and direction to the three-dimensional vehicle model. If the vehicle shoots or dies, a flash or burning hulk is rendered respectively.

The key to inferring new information from the post processor files is to compare the current event with the chronologically subsequent event for the same vehicle. From these comparisons, many useful pieces of information, such as the speed and direction of movement can be inferred. Currently, events in the JANUS post-processor files are listed chronologically [JANU 86]:

In order to compare two events for a single vehicle , there needs to be an easy way to locate the next event for a particular vehicle. Searching each of the five post-processor files for chronologically subsequent events is a memory intensive process that would quickly become time consuming and thus not useful. Therefore, an event file is created for each vehicle. Next, each of the JANUS post-processor files is read and the events pertaining to each individual vehicle is stored in its own file. Furthermore, a letter is appended to the head of each event to identify its event type (fire, move, etc.).

With the events for each vehicle stored in a single file, it is now easy to read an entire vehicle event file into memory, compare two chronological events for a vehicle, and infer some useful information about the vehicle's activities between listed events.

In general, there are three activities that a vehicle could possibly perform between listed events. It can be *halted*, *moving*, or *killed*. In order to present a realistic three-dimensional representation of these three activities, the following pieces of information are inferred:

- Orientation of the vehicle
- Orientation of the weapon (turret)
- Direction of movement (if moving)
- Speed

The *basic* speed and orientation of the vehicle and weapon are obtained by performing the simple calculations. The general mission of the vehicle is determined in order to gain a more refined estimate of its weapon's orientation and speed. These calculations are implemented using CLIPS Object-Oriented Language, (COOL) [GIAR 91]. COOL is used because the encapsulation of the information pertaining to one object makes it easy to manipulate the object as a whole.

There are several characteristics that suggest that a vehicle is in a defensive posture. The primary or tell-tale defensive characteristic is when a vehicle is travelling at a speed of less than 3 k.p.h. This speed tends to suggest that the vehicle is either on an extremely slow and deliberate offensive mission or it is actually sitting still (in a defensive position) for most of the time interval between the two listed events. If it moves to its next position at a more deliberate speed of about 13 -15 k.p.h., it is assumed that it is moving to an alternate defensive position. If this extremely slow speed is encountered, there are two other pieces of evidence that, if present, tend to confirm the defensive hypothesis. One of these additional pieces of evidence is whether the vehicle just completed firing a shot or will fire a shot in the future. This tends to be true because a vehicle shooting on the move would normally be travelling at a speed much in excess of 3 k.p.h. if it wants to survive. The second piece of evidence assumes that if a vehicle fires in the future, and if the distance to the next firing position is less than 100 meters, it is probably moving between fighting positions and hence is in a defensive posture. If all of the above conditions are met, then the program asserts that the vehicle is in the defense, and as a consequence halts the vehicle and orients it towards the enemy (towards the next target location). Then, the vehicle moves to its next event location at a set speed of 13 k.p.h., ensuring that it arrives at the next event location on time. There are certainly other heuristics that can suggest if a vehicle is in the defense or not, but they currently are not implemented.

A second tell-tale heuristic is whether a vehicle does not move for over five minutes. Not moving for over five minutes suggests that the vehicle is in a defensive posture -- viewing the engagement area. Then at a certain time it would move quickly to its next defensive position.

It is assumed that if a vehicle's events do not confirm the defensive heuristic then the vehicle is considered to be on an offensive mission. This is a simple but rather accurate hypothesis but does not exhaust the need for other inferred knowledge.

An anomaly that becomes very apparent in a three-dimensional world, that is not readily observed in the two-dimensional JANUS(A) simulation is, at times, a vehicle travels at a speed in excess of 45 m.p.h. Unless the vehicle is an aircraft, this is very unrealistic, and probably due to minor modifications in input by the user at the time of the original JANUS(A) run. This anomaly often occurs between the initial position for a vehicle and the first event listed in the post-processor files.

If these two conditions occur, then at start time, the vehicle is rendered at the position specified by the first movement rather than the location specified by the initial conditions. It is also given a speed of zero. If this *excessive speed anomaly* does not occur between the initial position and the first event, then the speed is smoothed.

The smoothing algorithm slows the speed of the vehicle to 25 k.p.h. This decrease in speed has a natural side effect. It will cause the vehicle to not be at the correct location when its next event is scheduled to occur (call this upcoming event, *A*). To compensate for this problem, a new MOVE event is created (event *B*) which will keep the vehicle moving at a speed of 25 k.p.h. to the location that was specified in event *A*. In order to get the vehicle back in synchronization with the remainder of the event list, the speed associated with movement from event *A* to its subsequent event is increased to allow the vehicle to arrive at the follow-on event's location on time.

As a rule, the orientation of the vehicle, if moving, is always in the direction of travel. If the vehicle is halted, the orientation depends on the following pieces of evidence:

- If the vehicle is firing during the current event then orient towards the target.
- If the vehicle is not firing but will fire in the future, orient the vehicle towards the next target location.
- If the vehicle is not firing and will not fire in the future, then use the vehicle orientation from the last event as the current orientation.
- If the vehicle is placed on the battlefield but does not have any events associated with it, then orient the vehicle in its initial view direction.

The heuristics for weapon orientation are similar to those used for the vehicle orientation with a few differences.

- If the vehicle is firing during the current event then orient the weapon towards the target.
- If the vehicle is not firing but will fire in the future, orient the weapon towards the next target location.
- If the vehicle is not firing, will not fire in the future, but did fire in the past then use the weapon orientation from the last firing event.
- If the vehicle never fires, then orient the weapon in the same direction as the vehicle.

Using the CLIPS Object Oriented Language (COOL), each of the events listed in a vehicle event file is read into one of the following objects: *MOVE, FIRE, KILL, ARTY* [GIAR 91]. After the entire file is read into memory, then each event is compared against the next event or an upcoming firing event. These comparisons generate facts that are stored in the form of (condition, <fact>). Then, based on the facts generated, CLIPS' inference engine will choose the appropriate rule to execute. These *rules* then call *methods* and *functions* which generate the data required for NPSNET [CLIP 91]. Once a rule is chosen and the NPSNET event data is generated, this data is written to a file specified for the current vehicle. Once the event file for all of the vehicles is evaluated, each of the files containing the heuristic data is combined into a single event list file. This file is then sorted chronologically. After sorting it is ready to be used by NPSNET as a script file.

## Real-Time Capability

The U.S. Army realized the advantages of running the JANUS(A) combat model on the popular and compact UNIX based workstations instead of on the cumbersome VAX/Tektronix systems currently in use. In August of

1991, TRAC-MTRY contracted with the Rand Corporation to develop a version of JANUS(A) that would operate on a Sun/UNIX workstation. A working prototype of JANUS(A) for UNIX was delivered in April 1992 [GUYT 92].

In the past, a network link between the traditional JANUS(A) model and NPSNET was not feasible because of the drastically different protocols and platforms. With the introduction of this new version of the JANUS(A) combat model, it finally became possible to construct this real-time network.

This new UNIX-based version of JANUS(A), which we call JANUS(X), does not change the "inner-workings" of the original combat model. The model itself is identical in both versions, in fact, the same FORTRAN code is used in both models. This reuse of the VAX-FORTRAN code is made possible by the Sun 4-FORTRAN compiler, which is capable of translating VAX data types and system calls into their UNIX equivalents [SUN 91].

The only viable difference between the two versions of JANUS is that the UNIX version bypasses all FORTRAN calls to the Tektronix screens, and replaces them with the 'C' programming language calls to the X-Windows environment [GUYT 92].

As mentioned above, the VAX-version of JANUS(A) is hindered by its antiquated hardware and non-networking capabilities. This version can only be displayed on Tektronix monitors, with the Blue force on one screen, and the Red force on another.

JANUS(X), on the other hand, has the added flexibility of being run on a Sun Workstation with the display piped to the same monitor, or any other workstation and monitor with X-Windows capability.

The communications between the Sun Workstation, running the JANUS(A) simulation, and the workstations rendering the two-dimensional images is accomplished using an Ethernet network. JANUS(X) creates messages and places them on the network as packets. These message packets are read by a listening workstation and rendered appropriately on its monitor [GUYT 92].

JANUS(X) executes two different programs at the same time. The main JANUS(X) program operates on one Sun workstation. This program executes all operations of the JANUS(A) combat model, initializes the network, and sends the message packets to the network. The second program runs on the workstation that will render the X-Windows screens. It captures the message packets that are on the network, then sends them to a sub-program which parses the messages and returns them to their original function formats. These functions then call the X-Windows library functions required to render the two-dimensional images on the monitor. Additionally, screen inputs to JANUS (e.g. mouse picks) are captured, recorded into a message packet, and sent back to the machine running the main JANUS model [GUYT 92].

The version of NPSNET used for this project uses NPS networking format. NPSNET and JANUS(X) use two different message sending protocols, and consequently, use different message formats. To accommodate both functions, messages are first sent from the JANUS(X) game using its own format. When these messages are read by the receiving IRIS workstation, they are changed to the NPSNET message format, and the appropriate three-dimensional graphics are displayed in real-time.

JANUS(X) sends a specific message to update the movement of a vehicle on one of the two-dimensional screens. Therefore, in the body of the JANUS(X) function to send this message, a special message destined for NPSNET is embedded. This message includes the side and unit number of the vehicle, its speed, orientation and current coordinates. Using the side and vehicle number as indices, NPSNET can adjust the location, speed and direction of the vehicle in the three-dimensional world.

JANUS(X) does not send any message which specifically identifies that a vehicle fired a shot. Therefore, a special function was written to write this message to NPSNET. This message sends the force side and vehicle identification number for both the shooter and the intended target. Upon receipt of this message, NPSNET will render a muzzle flash for the vehicle that fired. Also, a red or blue line (depending on the force side of the firer) is drawn from the shooter to the target. This line simulates the tracer action of a shot, while the color allows the observer a quick reference to who fired it.

Just as with vehicle shots, JANUS(X) does not send any message which specifically identifies that artillery was fired. So again, a function was written to send an artillery message to NPSNET. It sends the $x$ and $y$ coordinates where the artillery is currently landing. When this message is received by NPSNET, an artillery explosion is rendered at the coordinates indicated.

The death or destruction of a weapon system in the JANUS(X) combat model is simulated by simply removing its icon from the two-dimensional screen. To notify NPSNET of the destruction of a vehicle, a message to NPSNET was embedded in the function which deletes the icon. This new message sends the force side, vehicle identification number, grid coordinates, and the final orientation of the vehicle. NPSNET then simulates the destruction of a vehicle by halting it and displaying an orange flame.

## JANUS-3D as a Modeling Tool

JANUS(A) was used in the past as a modeling tool for the recent M1A2 main battle tank field tests, using the Army's Model-Test-Model (MTM) concept [BUND 91]. For the M1A2 tests, MTM began with pre-test modeling before the actual field test occurred. With the Fort Hunter-Liggett terrain database loaded in the JANUS(A) model, useful information was gleaned which helped design the field test site. Likewise, with the proper vehicle data, several battles were attempted to help decide upon the specific combat scenarios to run during the trials. After the test was completed, post-test modeling was done to compare against the data from the trials, in hopes to accredit JANUS(A) as a simulation modeling tool for MTM. If accredited, this would allow more operational testing and evaluation in the modeling environment, instead of the highly expensive field tests [PAUL 92].

JANUS-3D provides the combat weapons system developer the ability to easily visualize a new weapon system. The developer can develop appropriate tactics for the system, and test different weapon system specifications such as size, shape and positioning of optical devices prior to constructing any portion of the system.

JANUS-3D in the Pre-Test. The goal in the pre-test modeling is to use JANUS(A) to aid in the test design and help in recommending different combat scenarios. In most cases, the personnel conducting the pre-test modeling will be the leaders or users of the actual weapons system. Because they may have very little knowledge of JANUS(A), a user-friendly, interactive model, like JANUS-3D, is ideal for this purpose. After the terrain database is loaded into JANUS-3D, the unit leaders have the capability of driving every inch of the battlefield in their own vehicle. In the case of the M1A2 trials, each tank commander (four total) could sit behind his own workstation and drive, view, and shoot his own tank, via NPSNET's network capability. Offensive or defensive battles can be enacted time after time, against autonomous forces or against other manned workstations, to try and determine the best tactics and use of terrain, before the players have even seen the actual ground location. This helps save time and money in the overall test design, because the price of performing many pre-runs on an actual vehicle would be astronomical. Also, it provides excellent training and first hand knowledge of the area for the leaders and crews, which is a definite advantage going into the actual ground tests.

Besides serving as a training tool and scenario constructor, JANUS-3D also is an excellent tool to choose the areas for the field test. Test developers can position their vehicles in proposed defensive positions, check out their fields of view, and determine if the chosen piece of terrain is optimal for the weapon system's firing range. Likewise, proposed offensive routes can be driven to see if the vehicle has proper cover and concealment, adequate fields of view, and maneuverable terrain.

At the present time, the pre-test modeling accomplished by JANUS-3D cannot be input into the JANUS(A) model. Future research work with NPSNET: JANUS-3D will allow all activities accomplished in the three-dimensional world, such as vehicle positioning, movement, and firing, to affect the traditional JANUS(A) model. This feature would allow war-fighters and combat model analysts the capability to plan positions and routes in the three-dimensional world, then either JANUS(A) could run the actual scenario, or the operators could become participants in the 3D world, or a combination of both could be chosen.

A necessary requirement for the post-test phase is to replicate the actual field test site as accurately as possible within the model. Except for the resolution factors, the terrain in JANUS(A) cannot be changed. Ongoing work with NPSNET, on the other hand, has given the user the capability to dynamically change the terrain. At the present time, road craters, bridges (both erect and blown), and tank berms can be created and edited to fit the desires of the operator. Future work will allow tank ditches and tank positions to be excavated. This gives the analyst the flexibility to change the terrain to better match the actual test site.

In JANUS(A), if a tree or city lies within a grid cell, the entire grid cell is marked as having a tree or city, and has the respective height factor. For example, if the terrain database is using a 200 meter resolution map, and a grid cell is marked as having trees of height factor 12 meters, then the entire 200 x 200 meter grid cell would have a block of trees 12 meters high. Likewise, these height factors for the trees and cities do not come from the DMA terrain database, but are input by the user. Thus, if incorrect values are entered, or if the same values are used from database to database, these errors are not easily caught. However, erroneous height values become obvious in a three-dimensional world.

A better model of trees and cities is provided in JANUS-3D. By design, trees and buildings are stochastically placed in a marked grid cell, according to the density factor of the cell. The trees and cities are the accurate height, plus they display a more traditional shape. The random placement, coupled with the density of the trees, allows possible viewing under and around trees, which is more realistic in less densely packed forests.

However, if this is not acceptable, and an even better representation is demanded,
JANUS-3D has the capability of placing trees, cities, and most common man-made objects at the exact spot that they are located in the real world, with little difficulty. Thus, if aerial photos were available, or if someone had intimate knowledge of the area, the new objects for the terrain model could be accommodated.

Another requirement in the post-test phase is to replicate the vehicle actions as closely as possible. In the traditional JANUS(A) model, vehicles are represented as two-dimensional icons, and routes are planned by a series of straight lines connected by control nodes. An anomaly observed, after viewing several JANUS(A) scenarios in the three-dimensional mode, is that vehicles sometimes pass through one another, or actually park on top of one another (presumably at control points). These problems are caused both by human error upon input of the routes and nodes, and also because the model has no collision detection system. This is a serious problem, because the traditional model cannot accurately show that a battlefield may be too congested, or that there is insufficient room for one tank to pass a destroyed vehicle at a bridge crossing. JANUS-3D is an excellent tool to highlight these errors so that new routes or positions can be chosen.

In addition to movement errors, JANUS(A) also has problems with the field of view of stationary vehicles. JANUS(A) uses the last direction the vehicle travelled as the field of view. Thus, if a vehicle is in the defense, and withdraws to a subsequent battle position, his field of view could possibly be to the rear. Using JANUS-3D, the analyst will observe this problem, and can change the vehicles line of sight in the two-dimensional model. In future additions to JANUS-3D, the analyst will be able to affect this change from the three-dimensional model with the 'Space Ball'.

Line of sight calculations in JANUS(A) use a standard height factor for vegetation and vehicles, with each vehicle having the same factor. JANUS-3D allows the operator to place his view at the exact eye level for each vehicle, thus providing a more realistic view of the terrain. JANUS-3D shows problems with this calculation when vehicles are destroyed which cannot be seen in the three-dimensional world. Likewise, some vehicles in JANUS-3D have line of sight with enemy vehicles, but JANUS(A) does not show that they do.

## Conclusions

Two important results were achieved with this work. First, NPSNET was validated as a flexible, three-dimensional simulation platform for integration with traditional two-dimensional models. Second, it was shown that a traditional two-dimensional combat model can be successfully displayed in three-dimensions. This provides new life to systems that would otherwise rapidly become obsolete. What makes the second result even more noteworthy is that the three-dimensional display is achieved at a very low cost.

While this project proved that two-dimensional combat models can be displayed in three-dimensions, it only allows the user to be an "observer" of the simulation. The next logical step is to allow the observer to become an "operator" within the simulation. The goal is to have the actions of the operator within the three-dimensional world, be used as input to the two-dimensional model. This feature would allow war-fighters and combat model analysts the capability to test new ideas quickly and conduct multiple "what if" operations using the model.

## LIST OF REFERENCES


[BUND 91] Bundy, Dennis, "Model-Test-Model Using High Resolution Combat Models", TRAC-MTRY, Unpublished Paper, 1991.

[CLIP 91] Software Technology Branch, Lyndon B. Johnson Space Center, *CLIPS Reference Manual*, Version 5.0" JSC-22948, Jan 1991.

[FALB 92] Falby, John S., Zyda, Michael J., Pratt, David R., Mackey, Randall L., "NPSNET: A Networked Vehicle Simulation with Hierarchical Data Structures", currently under revision for *Computers & Graphics*.

[GIAR 89] Giarratano, Joseph C., and Riley, Gary, *Expert Systems, Principles and Programming*, PWS-KENT, Publishing Company, 1989.

[GIAR 91] Giarratano, Joseph C., *CLIPS User's Guide*, Information Systems Directorate, Software Technology Branch, Lyndon B. Johnson Space Center, Jan 1991.

[GUYT 92] Guyton, Jim, *JANUS(A) for Sun/Unix*, Rand Corporation, Los Angeles, CA, April 1992.

[JANU 86] U.S. Army TRADOC Analysis Command, WSMR, *JANUS(T) Documentation Manual*, June 1986.

[JANU 91] U.S. Army TRADOC Analysis Command, WSMR, *JANUS(A) Version 2.0 Information Letter*, March 1991.

[JANU 92] U.S. Army TRADOC Analysis Command, Leavenworth, *Janus Army, Single Model Serving All Domains*, Leavenworth, KS, 1992.

[MACK 91] Mackey, Randall L., *NPSNET: Hierarchical Data Structures for Real-Time Three-Dimensional Visual Simulations*, Master's Thesis, Naval Postgraduate School, Monterey, California, September 1991.

[PAUL 92] Paulo, Eugene P., Proctor, Michael D., Gorevin, Matthew L., Lathrop, Andrew, "Comparison of M1A1/M1A2 Battle Results Between JANUS(A) and an Operational Field Test", TRAC-MTRY, Unpublished Paper, 1992.

[SUN 91] Sun Microsystems INC, *Sun FORTRAN User's Guide*, February 1991.

[ZYDA 91] Zyda, Michael J. and Pratt, David R., "NPSNET: A 3D Visual Simulator for Virtual World Exploration and Experimentation", *1991 Society for Information Display International Symposium Digest of Technical Papers*, Volume XXII, May 1991, pp 361-364.

[ZYDA 91a]Zyda, M. J., Wilson, K. P., Pratt, D. R., Monahan, J. G., and Falby, John S., "NPSOFF: An Object Description Language for Supporting Virtual World Construction", currently under revision for *Computers & Graphics*.

[ZYDA 92] Zyda, Michael J, Pratt David R, Monahan Gregory, and Wilson, Kalin P; "NPSNET: Constructing a 3D Virtual World", *Symposium on 3D Graphics, '92 Proceedings*, April 1992, pp 147-156.