The Software Required for the Computer Generation of Virtual Environments

Michael J. Zyda*, David R. Pratt, John S. Falby, Chuck Lombardo and Kristen M. Kelleher
Naval Postgraduate School
Department of Computer Science
Monterey, California 93943-5100
zyda@trouble.cs.nps.navy.mil
*contact author

Abstract

The first phase of virtual world development has focused on the novel hardware (3D input and 3D output) and the graphics demo. The second phase of virtual worlds development will be to focus in on the more significant part of the problem, the software bed underlying "real" applications. The focus of this paper is on the software required to support large scale, networked, multi-party virtual environments. We discuss navigation (virtual camera view point control and its coupling to real-time, hidden surface elimination), interaction (software for constructing a dialogue from the inputs read from our devices and for applying that dialogue to display changes), communication (software for passing changes in the world model to other players on the network, and software for allowing the entry of previously undescribed players into the system), autonomy (software for playing autonomous agents in our virtual world against interactive players), scripting (software for recording, playing back and multi-tracking previous play against live or autonomous players, with autonomy provided for departures from the recorded script), and hypermedia integration (software for integrating hypermedia data - audio, compressed video, with embedded links - into our geometrically described virtual world). All of this software serves as the base for the fully-detailed, fully interactive, seamless environment of the third phase of virtual world development. We discuss the development of such software by describing how a real system, the NPSNET virtual world, is being constructed.

Keywords: virtual worlds, virtual world software.

The Phases of Virtual World Development

"Cool" graphics demos, and cable-entangled head-mounted displays on staggering, blinded individuals have been the signature of virtual reality research. We have seen an inordinate amount of attention on novel 3D input devices, e.g. data gloves, data suits, space balls, "flying mice", trackers, etc. We have also seen the same level of attention on novel 3D output devices, e.g. head-mounted displays, private eyes, VR caves, etc. The second phase of virtual worlds development is focusing on the more significant part of the problem, the software bed underlying "real" applications. It is the software that is the hard part BUT it is the head-mounted display and the data glove that currently get the attention. The third phase of virtual world development is the usage of the

software bed developed in the second phase for the construction of fully-detailed, fully interactive, seamless virtual environments. “Seamless” is the key concept, meaning that we can drive a vehicle across our terrain, stop in front of a building, get out of the vehicle, enter the building on foot, go up the stairs, enter a room and interact with items on a desktop. All without delay or hesitation in the system. To build seamless systems, there is a lot of software progress that must be made. Let’s begin by looking at the software bed to be constructed in the second phase of virtual world development.

Interaction Software

Interaction software is how we construct a dialogue from the inputs read from our 3D input devices and how we apply that dialogue to the system/application, and eventually display changes. There are two key parts to this software. The first is taking raw inputs from a non-standard 3D input device and figuring out what state that device is in. This gets done over and over again by everyone who buys one of these devices. Excellent technical papers and commercial sources are now available, quite worth examining before striking out on one’s own [Sturman, 1992], [Shaw, 1992], [Brill, 1993].

The second part of building interaction software is to turn the 3D device’s state information into a “dialogue” that is meaningful to our system/application. We need to be able to easily filter out erroneous or unlikely dialogues that might be generated by faulty data from our 3D input device. We need to generate messages to our virtual world system that execute some meaningful operation, i.e. flying, grabbing, etc. Again, there are some excellent papers on this [Robinett, 1992].

Interaction software is the first component we need for our virtual world systems and this is where everyone has been shining the light. Let’s look at the rest of the software necessary.

Navigation Software

Navigation software is how we move through our 3D virtual world. There are many component parts to this from the software side: 3D input device gesture interpretation (“gesture message from the input subsystem” to movement processing), virtual camera view point and view volume control, and hierarchical data structures for polygon flow minimization (to the graphics pipeline). Now all of these pieces of software are coupled! They all act in consort and in real-time to produce the next frame in a continuous series of frames, hopefully of coherent motion through our virtual world.

The 3D gesture interpretation is probably the easiest - most 3D input devices end up getting mapped to a familiar motion we already know how to do. The virtual camera control software is also starting to be well known, at least for flying through our virtual world [Ware, 1990].

Hierarchical data structures for polygon flow minimization are probably the least well-understood. A lot of people buy Silicon Graphics IRIS workstations and expect to be able to just blast all their polygons through the graphics pipeline! One million polygons per second is a large number, right? Why worry about it? This is a very common misconception! The problem we are trying to solve is that “reality is 80 million polygons per picture” [Catmull, 1993]. At 10 frames per second, this is some 800 million polygons per second.

So where are we today? We have workstations like the Reality Engine2 pumping out some 2 million polygons per second (flat shaded, non-textured). If we want textured scenes, we expect to run slower, say about 900K textured polygons per second. Now these are “marketing numbers” so we expect to see 25% of this, or 225,000 textured polygons per second. At 10 frames per second, this is 22,500 polygons per frame or 7,500 textured polygons at 30 frames per second. (For a reference point on what can be done with 7,500 polygons, we provide an NPSNET data point. The NPSNET system usually has some 1,200 polygons in its terrain displays, 150 to 200 polygons in each 3D vehicle on the terrain, and 17 polygons in each tree - there are 4,000 trees in one database. It is not hard to see that our 7,500 polygon budget is rapidly depleted.)

We either have to live with reduced complexity worlds or off-load some of the graphics work done in the pipeline onto the multiple CPUs of our workstations. And we have to do that polygon reduction in real-time and faster than just having it all sent through the graphics pipeline.

How hard is it to reduce polygon flow? This depends on what the virtual world consists of. This problem has historically been attacked on an application-specific basis and there is as yet no general solution. Solutions usually involve partitioning the polygon-defined world into volumes that can readily be checked for visibility by the “virtual world viewer” [Clark, 1976], [Airey, 1990a-b], [Funkhouser, 1992], [Teller, 1991]. There are many partitioning schemes - some of which work *only* if the world description does not change dynamically.

There is usually a second component to the polygon flow minimization effort and that is the “pixel coverage of the object modeled” question [Clark, 1976]. Once a volume’s objects have been determined to be in view, the secondary question is how many pixels will each object of that volume cover. If we find that the number of pixels covered by an object is small, then we can put out perhaps a reduced polygon count (low-resolution) version of that object. All of this results in additional software complexity, again software that must run in real-time.

Communications Software

Communications software is software for passing changes in the world model to other players on the network and software for allowing the entry of previously undescribed players into the system. It is quite easy to grow out of a single workstation when constructing a virtual world, espe-

cially if one expects multiple players in that world. When we move to a networked environment, we are beyond issues of graphics and interface software and into a much more complicated system.

When we move into a networked environment, we need to consider issues of database consistency more closely than we do in the single workstation world. We end up needing a standard message protocol between workstations that communicates changes to the world. For *small* systems, we need to make sure that all players on the network have the same world models and descriptions as time moves forward in the virtual environment "action".

What do we mean by a small number of players? We define low numbers of networked players to be between 250 - 500 players. The current SIMNET system looks like this and uses Ethernet & T1 links. Each node in the virtual world has a complete model of the world.

We define high numbers of networked players to be systems with 10,000 or more players. The Defense Modeling and Simulation Office (DMSO) wants this. For systems of this complexity, we can no longer afford to propagate complete models of the world but must start thinking about rolling in the world model just as aircraft simulators roll in terrain. Are many people thinking about this now? No, but perhaps we can see at least an abstraction that might be relevant in the work of Gelernter entitled "Mirror Worlds" [Gelernter, 1991]. In "Mirror Worlds", we have the notion of information tuples (think distributed blackboards), and tuple operations: publish, read and consume. Such an abstraction allows flexibility in communicating any type of information throughout a large, distributed system. This flexibility is just what we need in constructing large virtual worlds. While the abstraction looks appropriate, efficient and real-time implementations are an open research problem.

Autonomy & Scripting Software

We never have enough interactive players or workstations. We want software for playing autonomous agents in our virtual world against interactive players. The big issues here are what AI paradigms are we going to use to generate such players and how do we develop, in a uniform/standard way, a programmable mechanism for adding autonomous agents into our virtual world? Solutions to the autonomy software problem are how we are going to get beyond the "wow, we have 3D graphics working" phase in the VR community.

We need a capability for recording our interactions/play with our virtual world system. We need to be able to playback our interactions for later analysis. An additional goal of our scripting system is that we might like to be able to "multitrack" with our recordings, i.e. record another player in concert with a previously recorded player. We would like to be able to lay down some vehicle/player movement and then enter another vehicle for playing/driving in tandem with the original vehicle. The idea is not unlike the multitrack capabilities found in professional recording studios. The ulti-

mate goal for such a system is the smooth integration of pre-recorded tracks with interactive play, perhaps using autonomy and the ideas developed there for “departing” from the recorded script.

Hypermedia Integration

Hypermedia integration software is the addition of hypermedia data (audio, compressed video, with embedded links, text and still images) into our 3D geometrically described virtual world. The idea behind hypermedia integration is to combine virtual world technology with hypermedia technology by embedding hypermedia nodes in the virtual world. Hypermedia consists of non-sequential media grouped into nodes that are linked to other nodes. If we embed such nodes into say a building of our virtual world, the node can be accessed and audio or compressed video containing vital information on the layout, design or purpose of that building can be displayed, along with historical information. Such nodes will also allow us to make a search of all other nodes and find related objects elsewhere in the virtual world.

We also envision hyper-navigation. Hyper-navigation involves the use of nodes as markers that can be traveled between, either over the virtual terrain at accelerated speeds or over the hypermedia “links” that connect the nodes. Think of rabbit holes or portals to information or information places.

We also envision hypermedia authoring. In authoring mode, the computer drops nodes as a game is played. After the game, the player can travel along these nodes (which exist not only is space but also in time, appearing/disappearing as time passes) and watch a given player’s performance in the game.

Enough of examining virtual world software in general terms. Let’s now look at one virtual world that is serving as a prototype for developing the software for the second phase of virtual world construction.

The NPSNET Networked Virtual Environment

The NPSNET system is a low-cost, 3D visual simulator for virtual world exploration and experimentation.The goal of the project is to develop a basic virtual world shell that allows one to visit any area of the world for which a terrain database is available and to interact with other human or autonomous players found “in the system”. NPSNET is networked, i.e. we can put one interactive player at each graphics-capable workstation on the network (for however many workstations we have) and as many autonomous players as our computational and network resources allow.

The goal of the NPSNET project is to construct a low-cost visual simulator/virtual world explorer interoperable with the DARPA SIMNET system and the follow-on DIS networking standard. One of the tenants of the NPSNET project is to construct it on commercially available graphics

workstations, in particular the Silicon Graphics, Inc. IRIS workstation in all its incarnations. We wanted to have full control of the source code and not only build a SIMNET-compatible system but also several significant extensions. We wanted to experiment with some autonomous player ideas and improve the availability of the virtual world technology required to construct such a system. We wanted to build a virtual world system that could be put onto tape and shared with others without having to worry about whether it or the techniques involved were proprietary.

NPSNET Current Status

NPSNET is a family of virtual environment research systems, with variants numbered/named NPSNET 1 through 4, and Hyper-NPSNET. The core software technology advances with each system, with the newer versions supporting more advanced features.

NPSNET-4, the current developmental system, is being put together for a demonstration in the Tomorrow's Realities Gallery of SIGGRAPH '93. NPSNET-4 uses the Silicon Graphics, Inc. visual simulation toolkit Performer and is networked using the DIS 2.0.3 standard. The demonstration at SIGGRAPH '93 will contain a T1 link to the Defense Simulation Internet (DSI-net) and will have players at locations throughout the United States.

NPSNET maintains a 16km x 16km segment of terrain in memory and dynamically rolls in terrain as the driven vehicle moves about. NPSNET is capable of driving over any size area of terrain, with the only limitation being that of available disk space. NPSNET utilizes the parallel processing capabilities of multiple processor IRIS workstations to accomplish the terrain roll in without affecting the frame rate of the overall system. NPSNET will also run on the Indigo Elan IRIS, if texturing is turned off.

NPSNET Core Software Technology

We now turn to examine the core software technology developed to implement our system. By looking at the technology, we will have a good idea of the extent of the software bed required by the second phase of virtual world construction.

World Database Construction

To anyone who has constructed a 3D virtual world, it is clear that one of the major stumbling points is just getting the databases for that world. Such databases consist of the terrain model for the world, the 3D icons used to populate the world, the physics of the models used, etc. We utilize the SIMNET database for terrain, cultural features, and some 3D icons. We have added our own 3D icons to make NPSNET more interesting. We do not yet have a good 3D modeler to facilitate the construction of 3D icons.

Real-Time Scene Management

We could not just blast all the polygons describing our world through the graphics pipeline. Some of the terrain areas of interest are 300km by 200km, with postings every 125 meters. The terrain is covered with trees, buildings and other cultural features. There can be up to 500 vehicles, each comprised of between 150 to 200 polygons at full resolution. Because of this complexity, we spent a considerable amount of time on real-time scene management. In fact, in all of our virtual world and visual simulation efforts we have spent time on this [Zyda, 1993a]. For the NPSNET system, we had much more terrain data than CPU memory which added an additional level of complexity to our polygon minimization work. We used multiple resolution quadtrees, view volume computations to only display what was in view and generated full resolution data only for things near the viewer (driven vehicle) [Zyda, 1993a] [Zyda, 1992a], [Zyda, 1993c].

3D Object Modeling

NPSNET has a 3D object modeling language we call NPSGDL (NPS Graphics Description Language) [Zyda, 1992a] [Wilson, 1992] [Zyda, 1993c]. NPSGDL is an ascii-based language that we developed for storing our 3D models/icons for use in our 3D virtual world. We wanted to build a standard software interface for displaying and manipulating 3D models. NPSGDL is also our storage means for physical properties: polygons, materials, textures, lights, etc. NPSGDL also has support for animated variables, variables that can affect the appearance of a 3D object on a timed or free-running basis. NPSGDL is written in an object-oriented fashion so that each object instantiated can be in charge of maintaining its own changing appearance. Object-oriented programming is an essential element in minimizing the complexity of virtual world software.

Collision Detection

In NPSNET-1, vehicles could pass through objects (other vehicles, buildings, etc.). In the past year, collision detection has been added to NPSNET. A simplistic approach to the addition of collision detection is to have the computer make a constant search of all vehicles or objects in the world in order to determine if each vehicle is sharing three space with anything else. Since searching the entire world, including distant objects, is time-consuming, we have divided the world into a grid of squares, each square measuring $125m^2$. The computer checks everything labeled an “object” or “vehicle” within the grid-square occupied by the vehicle with an additional check to see if the perimeter of the vehicle crosses the perimeter of any other object or vehicle. If it finds that the vehicle is touching another object, it halts the vehicle’s progress in that direction and modifies the display according to a set of damage assessment rules. Collision detection increases the believability of the virtual world for obvious reasons, and is important for a simulation used for planning or training. If in a simulation, the user becomes used to being able to drive through things, that user

may not be aware of the exact size of the vehicle. This could lead to a dangerous and perhaps lethal situation when the trainee encounters a real situation. Realism is an important factor in virtual world development, but truthfulness is as important to simulations. If we teach people incorrectly in a simulator, we put that individual in danger and lose the usefulness of the simulation. A full discussion of the NPSNET collision detection and damage assessment algorithm can be found in [Zyda, 1993b].

Aural Cues for NPSNET

We are working on adding significant aural cues into NPSNET to further develop the "complete immersion" feeling for our players. The current system uses an Indigo Elan as a networked sound server. Workstations connected to NPSNET issue networked sound commands to the server via DIS packets. An Emu EMAX II sampler is connected to the Indigo server via an Apple RS-422 to MIDI converter. The sampler is capable of 16 parallel channels of sound and provides an aural dimension to NPSNET, with output to stereo Ramsas speakers at about 300 watts per channel. When an explosion occurs in NPSNET and the sound server is on, the office telephone four concrete walls removed cannot be heard. Maybe immersion is the wrong word. A follow-on project is to add 3D spatial sound via headphones attached to NPSNET's head-mounted displays. We are not pressing this at the moment as we have not yet found anyone who wants to wear a "face-sucker" more than 3 minutes at a time.

Physically-Based Modeling & Dynamic Terrain

Ballistic motion has been added to NPSNET. Ballistic motion applies the laws of physics to the world modeled in NPSNET. The most important of these effects is that of ballistic motion applied to projectiles. It is not enough for the projectiles to fly: they must fly in a manner consistent with physics so that the tactical information collected from the model is accurate. Physics has also been applied to the flight of aircraft in the latest version of NPSNET. Physically-based modeling of this degree can aid in the display of explosions, terrain modification and other transient events with a higher degree of realism than that available with the original NPSNET system.

Another aspect of NPSNET's physically based modeling is dynamic terrain. The idea of dynamic terrain is that the battle affects the environment. If a player knocks down a tree, the tree is represented throughout the network knocked down. These changes must be sent out across the network and reflected in each player's world view, even if that player joined the game after the tree was knocked down. In addition to trees, we also have berms, buildings and bridges that respond to dynamic changes. Such changes must be recorded so that the exercise can be resumed later should a hiatus occur. Physically-based modeling treats such graphics problems as products of physical

interactions. The information generated by these interactions is saved and sent over the network so that each player sees the same state of the world.

NPSNET-3 & Networking

NPSNET-3 was demonstrated in a networked test at the Institute for Defense Analysis in Alexandria, Virginia (Warbreaker/Zealous Pursuit Test, October through December 92). NPSNET-3 provided a 3D display for the test and interoperated with an F-15 simulator at McAir, St. Louis, another simulator at Williams AFB in Arizona, and several other simulators on the local IDA network (Patriot launcher, JSTARS C3 Station, etc.). The long distance networking was provided by the Defense Simulation Internet (DSI-net), which consists of T1 links at approximately 150 sites throughout the US.

The networking software architecture of NPSNET-3 is shown in Figure 1. The core simulator

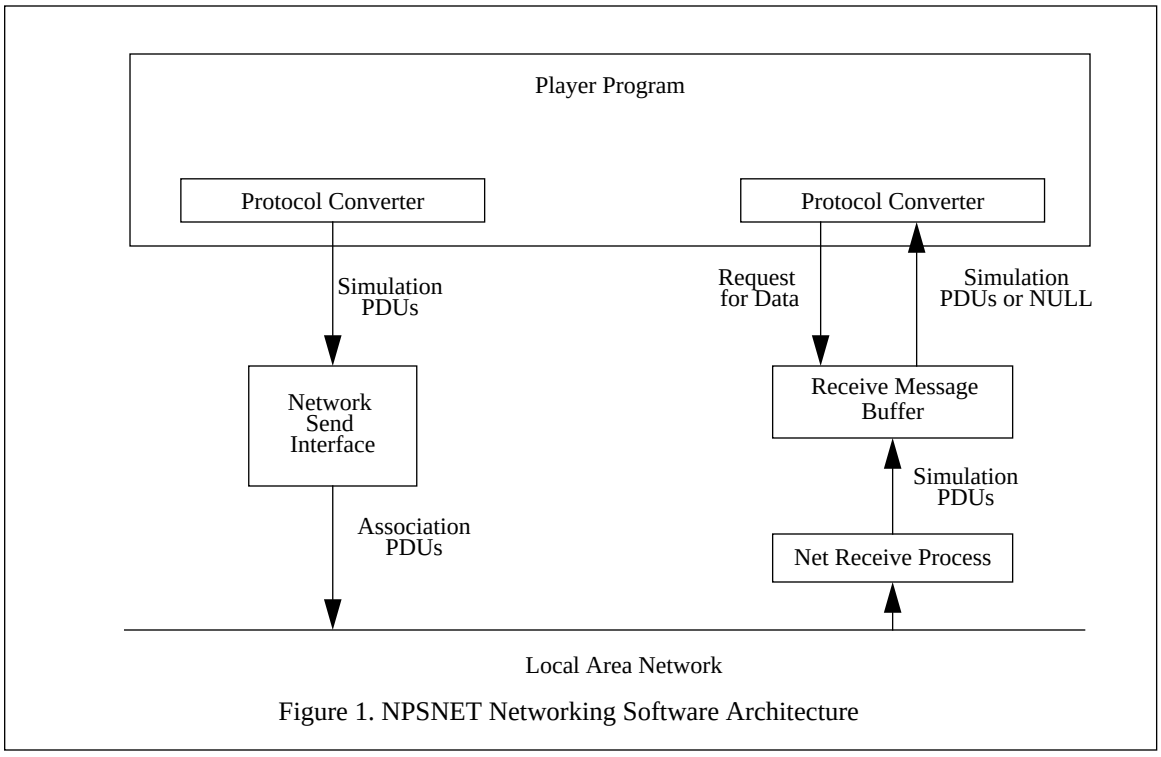

Figure 1. NPSNET Networking Software Architecture

communicates to the network via a protocol converter interface that sends/receives network packets asynchronously using both a “send thread” and a “receive thread”. This allows the graphics display rate to be maintained while data is read/written in separate, lightweight processes.

NPSNET Networking Performance

We have done two major tests measuring network performance. In our laboratory at the Naval Postgraduate School, we used a SIMNET Data Logger tape and SIMNET protocols to play an engagement at Fort Hunter-Liggett onto our local area network. In that test, we averaged approxi-

mately 100 messages per second for a period of 20 minutes coming from 270 different entities (interactive and autonomous players). The peak number of messages (Protocol Data Units -PDUs) was 210 PDUs in one second. The average packet length including the network header was 149 bytes. The simulation PDU traffic accounted for 50% of the network load but only 1% of the network bandwidth. The maximum one second burst was 2.3% of the bandwidth.

Our second network test was during the Warbreaker/Zealous Pursuit exercise. The goal of the exercise was to put players onto the DSI-net and see how the network performed and how the heterogeneous players interoperated.

The average SIMNET PDU rate for that test was 142 packets per second. The network encryption devices had a limit of approximately 170 PDUs per second before the devices slowed down, overflowed and crashed. The Warbreaker test was slowed down to accommodate this: only 100 entities were used, with only three of them being high performance aircraft flight simulators. Only ten transient munitions entities were active during the course of the exercise. Semi-automated forces (SAFOR) accounted for the other 70 entities on the network. The average network traffic accounted for 10% of the theoretical bandwidth.

We provide the following information on packets per second per vehicle as a reference for scalability of our virtual environments. High performance aircraft typically place 5 packets per second onto the network. Slow moving vehicles typically place 2 packets per second onto the network. All players, as per SIMNET protocol specification, place at least one packet every 5 seconds onto the network, just to report in that the player is still alive. Dead reckoning algorithms are used to minimize packets placed onto the network. Dead reckoning algorithms provide a predictive capability to each networked simulator. When each simulator can predict where a player is via constant speed, constant direction assumptions, no packet is placed onto the network. When a player's position cannot be predicted by the dead reckoning algorithms, that player is responsible for placing packets onto the network indicating new position and direction.

Future Networking & Systems Work

The current NPSNET system is capable of supporting some 500 players using Ethernet and T1 link technology. Like SIMNET, the software architecture of NPSNET is such that all players are recorded in the memories of all workstations on the network. As we begin to study how one handles 10,000 players, we must rethink the software architecture from both the networking and systems sides. This systems issue is a strong current focus, with considerations being given to the entire gamut of new computer architectures and computer networks predicted in the near future.

Hyper-NPSNET

Hyper-NPSNET is a real time interactive virtual environment with embedded multimedia capabilities. Within the system, there is the notion of an information anchor. These anchors are repositories for a variety of multimedia information. Hyper-NPSNET can have a large number of anchors each with hooks into video, audio, textual and graphics media. Audio in Hyper-NPSNET is stored as AIFC files. Video is stored as SGI Moviemaker files. The user navigates through the system and chooses either directly, or through proximity to an anchor, the multimedia information to view or hear.

In the current system, the user interface consists of multiple Motif panels to administer the information anchors and a SGI GLX Widget for rendering the virtual world. A 3D terrain database is read and used with texture information to create the ground of the virtual world. A multitude of buildings, trees, rocks, telephone poles and other miscellaneous objects populate the world. The end user typically loads an anchor database through the use of pull-down menus and pop-up windows. This database is created through the “Authoring” capabilities of the system (described below). Typically the user chooses to have the anchors visible at all times. This is a visual cue of where the information anchors are attached to the world. The user can trigger any of the available multimedia information by first selecting an anchor and then pressing one of the four buttons: audio, video, graphics or text on the main panel. To select an anchor, the user either selects it with the mouse directly in the 3D world, or chooses it off a list of available anchors displayed in the main control panel of the interface. Upon selection of an anchor, the main panel displays the current anchor name, type and coordinates. In addition, the current anchor is highlighted on the scrolling list of all available anchors. If the user selects an anchor off of the list, then the viewing point in the rendering window is transported to the location of the anchor in the 3D world. This is referred to as an instant aspect change. No matter how the anchor is selected, the user knows what kind of multimedia information is available for this anchor by noticing which of the audio, video, graphics and text buttons are sensitive.

Hyper-NPSNET Navigation

To gain access to all the anchors in the system, the user navigates through the world using either a Spaceball, Ascension Bird or standard 2D mouse. Through trials, it was determined that the most intuitive device and a device found on virtually every workstation was the 2D mouse. The Spaceball made it easy to move around, but difficult to pick anchors within the 3D world. The Ascension Bird had a disconcerting shakiness to the display that could not be overcome.

The user has a number of preferences that can be set through the use of the “Preferences” pop-up panel. Here the user can specify whether to fly around the world or drive on the terrain. If driv-

ing, the user steers left or right by moving the cursor left or right outside a small control square in the middle of the rendering window. To speed up, the user moves the mouse up, to slow down or go backwards, the user moves the mouse down. If flying, the heading is set with the left-right motion of the mouse, and the pitch is set using the up-down motion of the mouse. This leaves the speed to be controlled by some other means. The user can specify the flight speed in the panel. The user can also specify whether local anchors only are to be displayed. If local anchors only, the user can specify the range that defines what local is. The default value is 300 meters (the current terrain is 2 km by 2 km) meaning that only anchors within 300 meters of the current position of the user are displayed. The last thing the user can specify in the preferences pop-up is whether anchor information is displayed automatically as the user gets close to one of the anchors. If chosen, the user can specify the range used to trigger the multimedia information and can choose what information is automatically displayed. The default is Anchor Auto View off with a range of 20 meters and Audio media tagged. So if the user sets Anchor Auto View on and leaves other default values alone, then anchor audio tracks will play anytime the user gets within 20 meters of an anchor. This is known as *Audio Landmines*.

Hyper-NPSNET Authoring

Hyper-NPSNET can be used as an authoring tool for hypermedia. To build a new hypermedia database, the system is brought up without loading any anchor database. The user then creates all the anchors and attaches all the multimedia using the Anchor Editor Tool. For existing anchors, the editor is used to change any of the values for the anchor name, type, coordinates, orientation, audio track filename, video filename, graphics filename and text filename. For new anchors, the user simply enters all the new information about the new anchor and saves it to the system. An anchor will appear in the 3D location specified by the coordinates and will have all of the multimedia information available for viewing or hearing. Note that the audio, video, graphics and text files have already been created so the role of Hyper-NPSNET is a multimedia player not recorder at the moment. The integration of a real-time audio and video capture capability is a planned and not difficult extension.

Hyper-NPSNET Future Directions

Future directions for Hyper-NPSNET include allowing other than terrain (i.e. fixed) anchors. Useful ones that come to mind include *vehicle* anchors that are attached to moving vehicles. The user could query the vehicle for multimedia information about its design or capabilities. *Temporal* anchors may also be quite useful. Temporal anchors would only exist over some time range and would contain information relevant to the time associated with the anchor. Such anchors would allow the visibility of attached information only during the window of time specified with the anchor.

For example, a user of Hyper-NPSNET may only wish to view the video collected in March rather than have the display cluttered with the rest of the year's information anchors.

The system would also benefit from an advanced database system to manage all the hypermedia. This would be very useful for authoring and as an example, would allow the author user to browse all available video clips that match some wild card search parameters. Another possibility with such a system is the ability to visit all anchors that refer to a particular audio track.

Autonomous Players

The addition of autonomous players into NPSNET is critical to the realism of the simulation. When sufficient numbers of actual live players are unavailable or unaffordable, the simulation must provide interactive players. By utilizing a separate network "harness" process, which listens to the network and builds a state-of-the-world model, the simulation "grabs hold" of the unmanned vehicles and provides intelligent behavior models for them. We build in intelligent behaviors using the CLIPS expert system shell embedded in the network harness.

To make the autonomous tank forces act more like human players, we have given them the capability to work cooperatively. Three platoons of four tanks each can set up watch in different directions. If an enemy tank appears, the group assigns a tank (or a group of tanks) to dispatch them. If several enemies should appear coming from opposite directions, they can distribute their fire so that some of the tanks fire one way and the rest fire the other way. If more enemy tanks appear than are in that group of tanks, they are also equipped to call for reinforcements.

Another behavior given to some of the autonomous forces is the relay of information between a forward observation vehicle and several howitzers. For instance, the forward observer travels around the terrain. The vehicle is equipped with the ability to identify enemy vehicles. A determination is made about what the forward observer can see, based on the height of surrounding terrain. If all the surrounding terrain is flat, the forward observer can see four kilometers. If the terrain is mountainous, the determination is based on the average height of the incline. When the forward observer does identify a potential threat, the information pertaining to the enemy vehicle is sent to the howitzers. If they are in range, the howitzers fire on the sighted vehicle. The ability of these forces to work by themselves and actually contribute to the autonomous force effectiveness is a major step towards fully intelligent forces.

Some of the forces are equipped with a filter that allows them certain knowledge about the vehicles around them. These specially equipped vehicles can tell if a vehicle is friendly or not and whether they are within its range of fire or not. These vehicles can also prioritize targets so that important vehicles are eliminated before less valuable targets. These additions to autonomous force behavior make the vehicles more believable and more challenging to play against.

Autonomous behavior programming is completely separate from the NPSNET code, with a well-documented and standardized network interface. By doing this, we have made these behaviors easily modifiable, as well as low-cost.

We are still thinking about autonomy from scripts. There are a lot of circumstances where we would like to repeat a game “up to a certain point”. We then want to try things “differently”, perhaps by firing from a different weapons system. We need significant autonomy to do this and are channeling additional effort in this direction.

Conclusion

A fully interactive version of NPSNET is a continuing developmental effort. We are a long way from being done. At times there are several versions of NPSNET up and running, all prototypes testing out some new capability or feature. We have illustrated the software we are working on across the various versions as paradigmatic of the software bed necessary for the development of any virtual world. The very problems we are faced with are examples of why virtual worlds are slow to emerge. Virtual world software complexity and its interconnected nature are the primary holdups. We hope that as our efforts proceed, we will eventually succeed in completing a fully-detailed, fully interactive, seamless virtual world.

Acknowledgments

We are grateful to our sponsors who let us do whatever we want in exchange for unlimited demos. In particular, we thank ARPA/ASTO, the Defense Modeling and Simulation Office, US Army STRICOM, US Army TRAC-Monterey, and the Naval Postgraduate School Direct Funding Program for providing us funds with which to populate the NPS Graphics and Video Laboratory with the tools we need to carry out our “work”!

References

[Airey, 1990a] Airey, John Milligan “Increasing Update Rates in the Building Walkthrough System with Automatic Model-Space Subdivision and Potentially Visible Set Calculations,” UNC Technical Report TR90-027, July 1990 (Airey’s Ph.D. Thesis).

[Airey, 1990b] Airey, John M., Rohlf, John H. and Brooks, Frederick P. Jr. “Towards Image Realism with Interactive Update Rates in Complex Virtual Building Environments,” *Computer Graphics*, Vol. 24, No. 2, March 1990, pp. 41.

[Brill, 1993] Brill, Louis M. “Kicking the Tires of VR Software,” *Computer Graphics World*, June 1993, pp. 40 - 53.

[Catmull, 1993] Catmull, Ed, Carpenter, Loren and Cook, Rob - Private and public communication, January 1993. This quote is in reference to the number of polygons required to render reality, making certain assumptions about depth complexity and display resolution. This is dated pre-Renderman but the actual date is not known.

[Clark, 1976] Clark, James H. "Hierarchical Geometric Models for Visible Surface Algorithms," *Communications of the ACM*, Vol. 19, No. 10, October 1976, pp. 547-554.

[Funkhouser, 1992] Funkhouser, Thomas A., Sequin, Carlo H and Teller, Seth "Management of Large Amounts of Data in Interactive Building Walkthroughs," *Computer Graphics*, *Proceedings of the 1992 Symposium on Interactive 3D Graphics*, March 1992, pp. 11.

[Gelernter, 1991] Gelernter, David "Mirror Worlds: or the Day Software Puts the Universe in a Shoebox ... How It Will Happen and What It Will Mean," Oxford University Press, 1991.

[IST, 1991] Institute for Simulation and Training, "Protocol Data Units for Entity Information and Entity Interaction in a Distributed Interactive Simulation", Military Standard (DRAFT), IST-PD-90-2, Orlando, FL, September 1991.

[Pope, 1989] Pope, Arthur, "The SIMNET Network and Protocols", BBN Report No. 7102, BBN Systems and Technologies, Cambridge, MA, July 1989.

[Robinett, 1992] Robinett, Warren and Holloway, Richard "Implementation of Flying, Scaling and Grabbing in Virtual Worlds," *Computer Graphics*, *1992 Symposium on Interactive 3D Graphics*, March 1992, pp.189.

[Shaw, 1992] Shaw, Chris, Liang, Jiandong, Green, Mark and Sun, Yunqi "The Decoupled Simulation Model for Virtual Reality Systems," *Proceedings of CHI '92*, May 3 - 7, 1992, pp. 321-328.

[Sturman, 1992] Sturman, David Joel "Whole-Hand Input," MIT Ph.D. Thesis, Media Arts and Sciences Section, School of Architecture and Planning, February 1992.

[Teller, 1991] Teller, Seth J and Sequin, Carlo H. "Visibility Preprocessing for Interactive Walkthroughs," *Computer Graphics*, Vol. 25, No. 4, July 1991, pp. 61-69.

[Thorpe, 1987] Thorpe, Jack "The New Technology of Large Scale Simulator Networking: Implications for Mastering the Art of Warfighting," *Proceedings of the Ninth Interservice Industry Training Systems Conference*, November 1987.

[Ware, 1990] Ware, Colin and Osborne, Steven "Exploration and Virtual Camera Control in Virtual Three Dimensional Environments," *Computer Graphics*, Vol. 24, No. 2, March 1990, pp. 175.

[Wilson, 1992] Wilson, Kalin P., Zyda, Michael J., and Pratt, David R. “NPSGDL: An Object Oriented Graphics Description Language for Virtual World Application Support,” in the *Proceedings of the Third Eurographics Workshop on Object-Oriented Graphics,* Champery, Switzerland, 28 - 30 October 1992.

[Zyda, 1993a] Zyda, Michael J., Pratt, David R., Falby, John S. and Mackey, Randy L. “NPSNET: Hierarchical Data Structures for Real-Time Three-Dimensional Visual Simulation,” *Computers & Graphics,* Vol. 17, No. 1, 1993, pp. 65-69.

[Zyda, 1993b] Zyda, Michael J., Osborne, William D., Monahan, James G. and Pratt, David R. “NPSNET: Real-Time Collision Detection and Response,” *The Journal of Visualization and Computer Animation*, special issue on Simulation and Motion Control, Vol. 4, No. 1, January - March 1993, pp. 13-24.

[Zyda, 1993c] Zyda, Michael J., Wilson, Kalin P., Pratt, David R., and Falby, John S.Monahan, James G. “NPSOFF: An Object Description Language for Supporting Virtual World Construction,” *Computers & Graphics*, Vol. 17, No. 4, 1993.

[Zyda, 1992a] Zyda, Michael J., Monahan, James G. and Pratt, David R. “NPSNET: Physically-Based Modeling Enhancements to an Object File Format,” chapter in *Creating and Animating the Virtual World*, Editors: Nadia Magnenat Thalmann and Daniel Thalmann, Publisher: Springer-Verlag Tokyo, 1992, pp. 35-52.

[Zyda, 1992b] Zyda, Michael J., Pratt, David R., Monahan, James G. and Wilson, Kalin P. “NPSNET: Constructing a 3D Virtual World,” in *Computer Graphics, Special Issue on the 1992 Symposium on Interactive 3D Graphics*, MIT Media Laboratory, 29 March - 1 April 1992, pp. 147-156.