Editors: Lawrence Rosenblum and Michael Macedonia

# Training in Peacekeeping Operations Using Virtual Environments

R. Bowen Loftin, Mark W. Scerbo, Frederic D. McKenzie, and Jean M. Catanzaro

*Virginia Modeling, Analysis, and Simulation Center*

During the past 20 years, the US has engaged in two wars but has participated in roughly 30 major military operations. Although such operations have far more complex rules of engagement than large-scale wars, most military training still focuses on war fighting. Furthermore, because military operations today receive such intense news media scrutiny, the actions of even the most junior members of a military unit can profoundly shape world opinion, affecting the most senior levels of leadership, as we have recently seen.

Consider the current war in Iraq. During the war, a suicide bomber in a taxicab at a checkpoint near Najaf feigned engine trouble and then blew up the vehicle, killing four Coalition soldiers when they approached to investigate. Soon afterward, another vehicle approaching a checkpoint near Najaf failed to heed warnings to stop. Soldiers fired into it, killing seven women and children. Iraq remains a dangerous place. The Bush Administration declared the official war over nearly a year ago, but Coalition soldiers, military quarters, and supply lines continue to fall under attack.

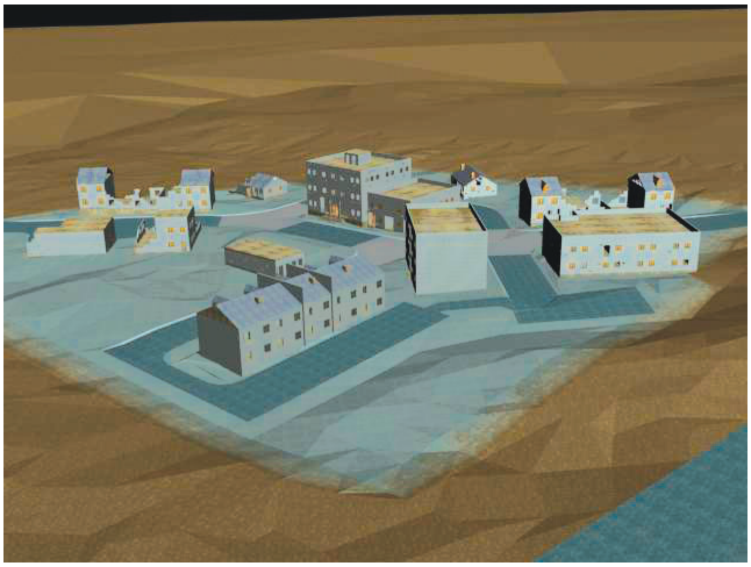
Such incidents make headlines and require military commanders to continually reevaluate their rules of engagement. Proper training of military personnel, at all levels, has never been more crucial. In this article, we describe the application of virtual environment technology to a novel and complex task—the military checkpoint. In the process, we also address differences between an immersive virtual environment training system and a desktop version.

## The training system

In developing the application, our primary objective was to reproduce the kind of experiences a military guard would encounter while standing watch. Thus, we created the task, setting, and virtual characters to match typical checkpoint conditions as closely as possible. The application recreates a checkpoint locale from the US Marine Corps training town in Quantico, Virginia. Figure 1 shows a bird's eye view of the terrain.

To create virtual human agents, we used Jack, a 3D modeling environment that supports high degree-of-freedom human models.[1] We selected the human models within Jack for this project because of the range of dynamic motion available, which includes utilities for locomotion; head and eye movement; and arm, leg, and all other joint movements. In Jack, the extent of motion of the human models is always within the physical constraints of selectable human body types, assuring us that the gestures and positions are within the realm of possibility given the particular human in a particular environment.

Because the process of manning a checkpoint can be a highly repetitive, mundane activity in which unusual events are rare, we developed two sets of training scenarios. The first contained general or neutral scenarios in which a vehicle approaches and stops at the checkpoint. The trainee inspects the vehicle and asks the driver—a virtual human intelligent agent—for identification. The trainee's partner (another virtual agent) provides cover for the trainee during the interaction. To more closely reproduce this activity's true conditions, each neutral interaction had to be unique, so we generated a pool of neutral scenarios that varied in vehi-


**1** Quantico terrain used for checkpoint location.

cle type and color as well as driver's gender and skin, hair, and shirt colors.

The second set of critical scenarios let the trainees exercise their judgment. The critical scenarios appeared at random intervals throughout the training session, unfolding without any cues to distinguish them from the neutral scenarios. For example, in one scenario, an ambulance arrives and the driver tells the trainee that he does not have time to go through the normal identification verification routine because he has an injured passenger. The trainee is expected to follow proper procedure and perform an identification check on both the driver and passenger, even if confronted with an urgent situation. Other scenarios required the trainee to be on the look out for a specific vehicle, detect a missing license plate, or identify the presence of contraband items, making the scenarios representative of the kinds of judgments a checkpoint guard must make.

## Immersive virtual environment

The first experiment took place in a fully immersive CAVE automatic virtual environment. With the images presented on two of the CAVE's 10 × 10-foot walls at a resolution of 1,280 × 1,024 pixels, we used LCD CrystalEyes stereo shutter glasses to view the images stereoscopically. Positional tracking was provided through Ascension Technology's Flock of Birds software, a six degrees-of-freedom tracker with a single head sensor attached to the CrystalEyes glasses. Three main computing systems connected through a 100-megabits-per-second network switch:

- An SGI Onyx 2 computer displayed the application in the CAVE, provided the sound playback, and read the information from the tracking devices. This computer uses Titan Corporation's VrTool which includes VrGui and VrSpeech, VRCO's TrackD, Jack, Python opensource scripting language, SGI Open Inventor, and IRIX 6.5 operating system .
- An SGI O2 computer served as the experiment's main console. From here, users could launch the application and operate override controls during the simulation. This computer uses IRIX 6.5, SGI Motif, and SGI.
- A PC computer also hosted the voice-recognition software, communicating the information to the SGI ONYX trough a network socket. This computer uses Microsoft Windows 2000, IBM ViaVoice, and VrSpeech.

The "Porting to PCs" sidebar discusses our efforts to move the application to less expensive PC-based systems.

## Porting to PCs

Because expensive graphics workstations are becoming less popular and new PC-based VR environments are emerging, we ported our checkpoint training system to such an environment using the Quantum3D ObsidianNV Rack for the PC platform. The Quantum3D ObsidianNV Rack has four interlocked PCs that can drive four channels simultaneously. It's suited for a multidisplay VR environment. Currently, the system drives separate flat panel displays using distributed Vega—MultiGen-Paradigm's software environment for the creation and deployment of real-time virtual reality applications—to drive the multiple scenes. However, the system clearly can directly drive a CAVE as well. Of the six components that were necessary, we only used two in the port.

Scene graph construction and rendering was done with Titan Corporation's VrTool. There are two scene graphs, one in VrTool and the other in Jack. The Vega platform supports distributed simulation and real-time rendering. Porting from VrTool to Vega was fairly straightforward; they both follow the same design philosophy, both having a user-friendly, top-level configuration editor working in conjunction with a lower-level API for complex programming. Vega has some nice, simple programming examples that are well suited for beginners and people without significant technical backgrounds. Most surprisingly, the Vega documentation is good in parts but in others, particularly for the more complex areas, whole sections had been left out. For example, nowhere does the documentation explain how to retrieve information on more than one isector segment from a collision detection volume.

Porting from Jack to Boston Dynamics DI-Guy was more difficult. We decided not to continue using the Jack toolkit in our PC port for four reasons:

- the licenses for it were expensive,
- the software was not mature,
- it had poor documentation, and
- it was poorly supported.

In the PC port, we used DI-Guy instead, choosing that option because of its high-quality human models, and its ability to generate high-fidelity animated actions.

LWNets are a component of the Jack toolkit used to control the behaviors of the virtual humans. As Jack was not ported to the PC, we developed a decision and logic network using states and conditions based on time and location.

Porting VRCO's TrackD was by far the easiest. Our experience reflects on TrackD's well considered design and multiplatform support. Essentially, the code was exactly the same for both aspects.

Porting the windowing system was also easy. By using Vega, it was not necessary to create the graphics display windows, as these were created automatically. We used standard win32 API calls for human–computer input.

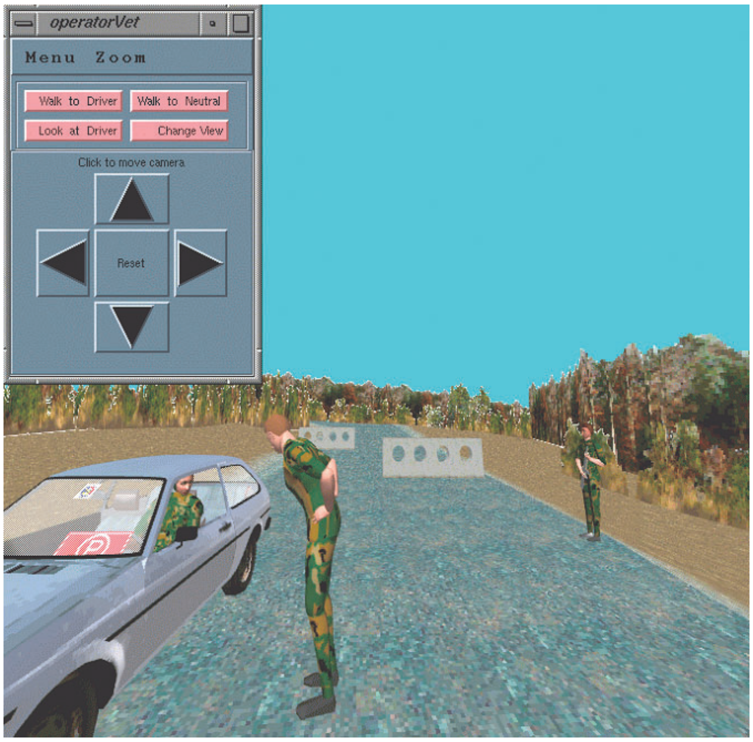
Finally, no porting was needed for the speech recognition because the original was already designed to run on a windows PC laptop.

## Interactivity

Head tracking and voice recognition were the two human–computer inputs available to the trainee. The trainee's tracking position fed into our virtual environments for training (VET) system, which in turn updated the VR automobile driver's head and eye orientation. As a result, the virtual humans commanded a life-like presence in the environment.

Trainees also wore wireless microphone headsets to issue voice commands to the PC running IBM's ViaVoice software. This software recognized any valid words or sentences according to the grammar that we developed, transmitting the results via network socket communication to the SGI computer running the main program. The main program then checked words and sentences to

2 Trainee interacting with a driver in the CAVE environment.



3 GUI created for the desktop system.

determine if they were valid commands; if so, the application set a corresponding event.

Scenarios were scripted using a simple text configuration file—essentially a large table of the elements that needed to be present in each scenario. Elements included vehicle type, driver's gender, driver's ID type, passenger's ID type, and sticker or pass information. For each scenario, the program initialized according to these parameters. Other parts of the scripting, such as virtual driver responses, used the current scenario number as an index to a large bank of over 1,500 sound and voice files. The application created background and other supplemental audio sounds—including gunfire, airplane flybys, wind, and the approach of a car on a gravel surface—using a combination of existing sound samples and environmental sound recordings.

The combination of interactive elements provided a significant sense of immersion and a level of interactivity believed to be unparalleled when developed last year. Figure 2 shows the system in use. In this particular scenario, the driver suspiciously averts eye contact to check his car rearview mirror and his cohorts in the vehicle behind. If the trainee neglects to signal his virtual partner, the driver in the vehicle behind would produce a gun and fire.

## Desktop virtual environment implementation

We created a different interface for a desktop version that lets trainees navigate and inspect the vehicles. There were two main differences in the desktop system's hardware:

- We replaced the SGI ONYX2 Image Generator with an SGI Octane desktop computer. Also, the desktop version does not display images stereoscopically, instead presenting them on an 18-inch Sony flat panel display. Therefore, no shutter glasses were used.
- There was no positional tracking with the desktop system. Instead, we developed a new interface to let participants navigate within the scene and inspect the vehicles.

The trainee's graphical user interface sits over the rendered scenarios as Figure 3 shows. The GUI has two drop-down menu options in the toolbar: menu (play, pause, speech toggle, or quit) and zoom (zoom in, zoom out, or zoom reset). Pushbuttons provide features such as "walk to driver," "walk to neutral," "look at driver," and "change view." The first two let the trainee avatar walk to a location that is offset from the driver and walk back to a predetermined neutral location. The "look at driver" feature bends the avatar's torso so that the driver comes within his line of sight. "Change view" simply toggles between the original camera view (overlooking the checkpoint scene) and the avatar's view. In addition, the GUI has buttons (large directional triangles) to pan the camera left, right, up, and down, and also to reset it to a default position. Using these buttons, a trainee could inspect a vehicle's front and back seats.

Trainees used the GUI to navigate within the scenarios and inspect the vehicles. The experimenter used a second GUI to control the driver and passenger responses. The experimenter's GUI had all the capabilities of the trainee's GUI for accomplishing the appropriate adjustments, prompting, and demonstrating. Driver and passenger control was available in the form of button pushes for triggered responses such as, "Yes sir," "I don't understand," and "Okay."

## Computing elements

Most of the source code concerns setting up a world state and then rendering the virtual world according to this state. It has a main loop that cycles through at approximately every 10 milliseconds to update this state. Included are such tasks as collecting user input, calculating the objects' positions and orientations in the virtual world, and outputting to a number of displays.

Jack uses an abstract factory design pattern to generate virtual humans. This pattern creates families of related objects without specifying concrete classes. Essentially, Jack lets users designate procedures that are responsible for creating, updating, and deleting components of the virtual humans, such as the limb segments and joints. The class we designed for creating each virtual human derives from Jack's behavior base class. This class contains a collection of variables used throughout the program for scripting and other purposes, and also contains several figure objects. Each figure is a special object-oriented class that encapsulates the human's geometry and has the procedures that pass to Jack's abstract factory and serve to convert the geometry into virtual humans in the scene graph.

We used VrTool for scene graph construction and rendering. There are two scene graphs, one in VrTool and the other in Jack. VrTool's scene graph is what is actually rendered on the screen, whereas Jack's serves internally for dynamic character animation calculations. A special type of inventor file called a scene file initializes VrTool when it starts up. We can set all of these using the top-level configuration editor, VrGui, which is similar to MultiGen-Paradigm Vega's Lynx. Any number of other visualizers would have also been appropriate in place of VrTool's, such as Vega's Lynx or even straight OpenGL.

## Performance

Undergraduate students from Old Dominion University trained in either the system's immersive virtual environment or desktop versions.[2] We selected this population because these students are representative of the type of individual who would likely be assigned to military guard duty. We gave them background information on their role and responsibilities, told them to assess the vehicles and occupants, and asked them render decisions as to whether vehicles could enter. Half performed a 45-minute shift, received feedback on that shift, and then performed a second 45-minute shift. The remaining participants performed only a single session. We compared the performance of these participants to that of another group who only performed a single session using the second session's scenarios.

We assessed performance by the total number of errors each participant made on each scenario type. As expected, more errors were committed on the critical scenarios. Also, the trainees who received two sessions made about two-thirds fewer errors on their second session. Table 1 shows results for critical scenarios.

A second analysis compared performance between the participants in group 2 and group 1 in their second session. The group 1 trainees also made about two-thirds fewer errors in their second session when compared to group 2 participants. The better performance of participants in group 1, session 2, over those of group 2 indicates that they benefited from their training experience.

Table 1 also shows performance with the desktop system. The overall pattern of results was similar across systems. Those individuals who participated in two sessions showed marked improvement in their second session and performed better than another group of individuals who only participated in a single session. More importantly, the overall performance levels for all participants in all conditions were better with the immersive environment than with the desktop system. This difference might be attributable to the CAVE experience and the benefits of immersive virtual environment technology. Such results indicate that CAVE technology has great potential as a training tool with applicability to the development of decision aids and selection tools.

**Table 1. Mean Number of Errors for Critical Scenarios in the Immersive and Desktop Studies.***

| | Immersive VE | | Desktop System | |
|---|---|---|---|---|
| Session | Group 1 | Group 2 | Group 1 | Group 2 |
| 1 | 6.36 (6.0) | 5.47 (3.41) | 7.43 (3.26) | 8.14 (5.24) |
| 2 | 1.87 (4.51) | | 3.0 (2.71) | |

*Standard deviations appear in parentheses.

The objective data indicate that the participants responded well to the virtual environment. Most participants acclimated quickly to the environment, became accustomed to the methods of interaction, and interacted with virtual objects rather naturally. On a more subjective level, evidence suggested that the participants were immersed in the task: We observed some individuals using hand gestures to motion cars to pull up to the gate and others reaching out to try and hold the ID card presented by the driver.

## Conclusions

These results indicate that individuals can benefit from training in a virtual environment that places greater emphasis on social interaction skills. Individuals with little or no military training were able to learn some of the fundamentals for performing checkpoint duty in an experiential context. These findings should encourage those in the development community to continue to improve and refine the technology required for this class of virtual environments. Additional work on modeling body gestures, facial expressions, and voice recognition in real-time simulations is needed to develop training for more complex human interactions. ■

### References

1. N.I. Badler et al., "Parameterized Action Representation for Virtual Human Agents," *Embodied Conversational Agents*, J. Cassell, ed., MIT Press, 2000, pp. 256-284.
2. B. Loftin et al., *Virtual Environments for Training: Final Report*, Office of Naval Research, under Grant N00014-95-1-1044, 30 Dec. 2002.

*Readers may contact the department editors at* rosenblu@ait.nrl.navy.mil *or michael_macedonia@stricom.army.ml.*

*Readers may contact Bowen Loftin at the Virginia Modeling, Analysis, and Simulation Center, Old Dominion University, 1 Old Dominion Univ.,, Norfolk, VA 23529;* bloftin@odu.edu.