Simulating Participant Training Data to Test Mixed-Reality Training Systems

Ken Kopecky, Eliot Winer

Iowa State University

Ames, Iowa

kennyk@iastate.edu**,** ewiner@iastate.edu

Julio de la Cruz

Army Research Laboratory HRED-STTC

Orlando, FL

Julio.Delacruz@us.army.mil

ABSTRACT

As simulation-based, mixed-reality, and virtual reality training systems are more widely adopted in the military, the process of verification and validation (V&V) for these systems becomes similarly more complicated and time-consuming. It is critical to verify and validate these simulation-based training systems so they operate properly and as expected. Often, live trainees are brought in, hardware set up, and different configurations of a system tested as part of the V&V process. No actual training has occurred. In much the way that these simulation systems act as a substitute for live action, the subsystems and trainees that use training systems can also be replaced with simulations to dramatically speed-up V&V. This paper examines the potential for replacing live trainees and hardware with virtual simulations in a mixed-reality training environment for the purposes of V&V of a simulation training system.

A case study is presented, composed of a method that allows multiple tracking systems, from different vendors, to be combined into a single system. The system, used in a large mixed-reality training environment, allows different aspects of the physical layout to be tracked depending on the training being performed. In order to test the system's robustness, virtual tracking data was generated, having been calibrated from actual tracked entities, to test metrics including positional error correction and data throughput capability. Comparing this data with results obtained using real tracking hardware allowed the development of models to predict the system's behavior in new situations, such as the introduction of a new tracking system, or introducing a second tracked space to the training simulation. Using simulated tracking data, errors were identified in the system without the need for testing with humans or additional equipment. Finally, the virtual data was used to test the simulation itself, to ensure it would handle the data requirements encountered during actual training. Results indicated that simulated data can be used to test the various factors necessary for V&V of a simulation-based training system. The training system behavior was the same whether actual or simulated data was used. The use of simulated data allowed scenarios to be tested without the need to bring in additional human and equipment resources.

ABOUT THE AUTHORS

Ken Kopecky is a Ph.D. candidate at Iowa State University's Virtual Reality Applications Center (VRAC). His research areas include virtual and mixed reality, game development, and middleware. Ken is currently developing software that enables people to easily use the hardware and software available for mixed reality research with their own applications.

Eliot Winer, Ph.D., is the associate director of VRAC and associate professor of mechanical engineering at Iowa State University. He recently led research developing and integrating four virtual and three live environments in a simultaneous capability demonstration for the Air Force Office of Scientific Research. He is currently leading an effort to develop a next-generation, rapidly reconfigurable, mixed-reality virtual and constructive training environment for the US Army. Dr. Winer has over 15 years experience working in virtual reality and 3D computer graphics technologies.

Julio de la Cruz is the Chief Engineer for Synthetic Environments Research at the Simulation and Training Technology Center in Orlando, FL and is responsible for the research and development of applied simulation technologies for learning, training, testing and mission rehearsal. He has a B.S. degree in Electrical Engineering from the City College of New York and a M.S. in Industrial Engineering from Texas A&M University. He is a graduate of the U.S. Army School of Engineering & Logistics and alumnus of the Advanced Program Managers Course at the Defense Systems Management College (DSMC).

Simulating Participant Training Data to Test Mixed-Reality Training Systems

Ken Kopecky, Eliot Winer

Iowa State University

Ames, Iowa

kennyk@iastate.edu**,** ewiner@iastate.edu

Julio de la Cruz

Army Research Laboratory HRED-STTC

Orlando, FL

Julio.Delacruz@us.army.mil

INTRODUCTION

Modern Warfighter training is becoming increasingly focused on simulations. A simulation can provide an immersive and realistic training experience for participants at a significantly lower cost than live training. However, a limiting factor in the effectiveness of a simulation is the set of hardware and software systems on which it is dependent. The process of calibrating and maintaining a mixed reality training system (MRTS) requires a great deal of time and manpower. In order to test the system in a way that closely mimics real use, live trainees must be brought in and all the various aspects of the system must be set up and initialized. This requires significant time, money, and manpower that could otherwise be devoted to actual training. While data from real human testing is very valuable, it is difficult to have humans perform the exact sequence of events repeatedly, a process often necessary for solving hardware or software problems that arise. Recording and playback of this human data is possible, and is recognized as a valuable part of software testing, as evidenced by tools like Apple's User Interface recording tool (Apple, 2014). Playback and testing with recorded data can even be carried out in faster-than-real-time, speeding up the development process. But recordings are only useful as long as the training system doesn't change very much. For example, if the layout of a room used in a training scenario changes, a record of trainee movement that was previously valid might now have the trainee moving through a wall. Testing methodologies are needed that remain valid even if significant changes are made to the training system. Warfighter training is amongst the most highly fluid situations. Training must constantly be adapted to new threats and skills acquisition to allow a warfighter to remain effective in the field. To that end, we will discuss the use of simulated input data as a way of addressing some of the limitations of real-time human input data and recorded input data.

Simulated inputs are those that are generated by software to mimic the actions of humans or other equipment in a training system. These actions are converted to the input signals of the MRTS, such as position tracking data, trigger pulls on simulated weapons, or auditory commands. A major strength of simulated input is it can adjust or be adjusted to match changes in the system being tested faster and more cost effectively as compared to recording actual human and equipment input in the event a training scenario changes. In certain industries, this is common practice. For example, simulating data from a WiFi network may entail replicating network data packets from multiple access points and network switches. This allows access points or network switches to be tested in a wide range of operating conditions without having to physically install them.

Through simulated inputs, data can be fed to a simulation-based training environment or MRTS faster than is currently possible with existing hardware. As future advances realize these rates, the training environment can be developed to accommodate this in real-time. Lastly, simulated input can be automated. An entire battery of tests can be performed at a single keystroke without the choreography needed to record human and equipment input in traditional training systems testing. This paper will discuss the use of simulated data to test and improve MetaTracker, a system for combining and disseminating 3D motion tracking data to training/simulation applications. In particular, it will highlight ways in which simulated data greatly sped up the process of repeatedly testing MetaTracker's data processing functions and allowed high-load testing that would otherwise not have been possible.

BACKGROUND

This section will talk briefly about some of the types of software testing, and will then focus on simulations and the types of data they produce.

Software Testing

Software testing is an extremely broad field. There are many types of tests that can be used during development, and several ways of categorizing them. Some tests are more suited to automation than others. Usability testing, for example, is the process of observing and/or recording a person while they use an application to determine the level of quality of its user interface. Factors such as number of errors made and task completion time are measured (Chisnell 2009). Usability testing, by definition, requires human interaction, and clearly cannot be automated. Functional testing involves making sure individual components of a system or application perform their tasks properly, such as checking if a simulated bullet impacts the correct object in a pre-defined situation (Pan 1999). Non-functional testing, on the other hand, serves to test the readiness and quality of a system or several of its related components(Pal 2014). Many types of testing fall under the umbrella of non-functional testing, such as scalability and reliability testing. A scalability test, for example, could involve setting a crowd simulation to simulate the behavior of a hundred times as many people as its anticipated use case would normally have, then measuring its performance and stability. Both functional and non-functional testing can often be automated to speed up the development process.

Simulations for Training

Simulations are widely used in training. For example, flight simulators allow pilots to become skilled at operating a fighter jet without ever entering a actual cockpit. For training Warfighters in urban peace-keeping scenarios, crowd simulations can approximate the behavior of a group of civilians, with constructive humans responding to the actions of the trainees(Zhou et al., 2010). Game engines in particular include many types of simulations. Bohemia Interactive Simulation's VBS2 platform (BohemiaInteractive, 2013) features a physics engine and AI (artificial intelligence) control of simulation entities. These two features are combined within VBS2 to allow virtual vehicles and dismounted soldiers to realistically interact with terrain, buildings, and other objects. Similar capabilities are found in Crytek's CryEngine(CRYTEK, 2013) as well as other game engines.

For more hands-on training than can be provided by a desktop computer game, virtual and mixed reality training systems are important technologies. Mixed reality training simulations generally involve live trainees moving around a physical set. (Pollock, Winer, Gilbert, & de La Cruz, 2012) Computer displays act as viewports into the virtual world, and participants use simulated weapons to fire upon virtual and constructive enemy combatants. Additional interaction methods may include voice recognition and other digital input devices. Mixed reality training simulations require a wide variety of components to work together including computer displays, specialized input devices, tracking systems, structures, props, and, most importantly, people. All must be in their correct places and configurations, and these configurations must match what the training simulation is expecting. For example, if a stereo screen is intended to sit in the window of a structure and act as a view into the virtual world, the computer generating images for it must know its position and, if perspective-correct stereo is needed, the position of the person viewing it to generate the proper view.

Trainees and their simulated weapons often need to have their positions tracked in mixed reality training. 3D tracking allows computers controlling the simulation to react properly to the movements of participants and the positions and orientations of their weapons. There are a number of different types of technology used for tracking, such as optical, magnetic, radio frequency, and even hybrid systems that use two or more technologies to improve tracking accuracy. The sorts of environments used in mixed reality training can pose challenges for tracking systems. A large number of objects must often be tracked, including every trainee, weapons, and other objects whose positions are relevant to the scenario. Moving from room to room can cause optical trackers to be occluded, or exceed the working range of optical and other technologies. If 3D tracking technologies are not used, weapon aiming can be accomplished through laser detection, such as that used in the Laser Shot weapons training system (Martin, 2013).

One current example of mixed reality training, the Virtual Convoy Simulator (MacLeod 2013) shown in Figure 1 allows trainees to become immersed in the deserts of Afghanistan as they embark on simulated convoy operations using four life-size HUMVEEs with simulated weapons, both mounted and dismounted.

Figure 1. The Virtual Convoy Simulator(MacLeod, 2013)

Simulations operate by applying heuristics, constraints, physics principles, and other rules to a model of a situation. The more rules are applied to a simulation, the more rigorous it becomes. A low-fidelity physics simulator simulating a falling stack of blocks might take factors such as gravity, conservation of angular momentum, Newtonian friction, and collision elasticity into account while ignoring things like air resistance and deformation of the blocks themselves due to impact. The resulting output would likely be very believable, but not physically correct. A simulation such as this is appropriate for cases where the correctness of the result is less important than its plausibility, such as in a first-person shooter video game. Higher-fidelity simulations, such as those involving FEA (finite element analysis) techniques, can more accurately predict real-world outcomes, but come at a cost, such as more complex implementations, an increased number of input parameters, and a longer simulation time.

The use of simulations for testing physical systems has been well-documented and is a key concept in Simulation-Based Acquisition of military equipment. It is described extensively by Lutz, et. al., (Lutz et al., 2013) in a use case for testing the Sense and Avoid (SAA) systems for the Navy's Triton unmanned aircraft. The team addressed the use of various types of simulations to model the airspace in which the Triton would fly. The systems used combined simulations of differing levels of fidelity, only using high-fidelity models when it was necessary for the accuracy of the testing. In the cases requiring the highest accuracy, humans were used to operate simulators, rather than purely constructive entities.

METHOD DEVELOPMENT

For this project, simulated data was created to test and evaluate MetaTracker, a system for combining and abstracting 3D tracking information in a mixed-reality training system. MetaTracker is designed to simplify the process of getting positional data from multiple 3D tracking systems to the simulation applications that use it, without the simulations having to be aware of the individual tracking systems or their drivers. A diagram of MetaTracker and its components is shown in Figure 2.

MetaTracker is made of three major components: The source applications, the server, and the clients. Each source application connects directly to a tracking system and receives positional and rotational data from it. This data is immediately forwarded, via UDP (User Datagram Protocol) packets to the MetaTracker server, which can combine data from multiple tracking systems and make them act as a single system. MetaTracker's server, seen in Figure 3, visualizes tracking data. The server may also perform manipulations to the data, such as enforcing degrees of freedom for specified objects. For example, a faux concrete barrier used in a simulation may only have three degrees of freedom (assuming it can't be knocked over). It can move along the horizontal plane and rotate about a horizontal

axis. The Clamp To Ground option of the MetaTracker server can be used to eliminate any movement or rotation along the remaining axes, based on the assumption they're due to tracking system error. Finally, the server sends the tracking data to each client application. A client application is any that uses the 3D positional data, such as a training simulation or tracking data recorder. For example, an immersive application for live trainees, that is part of a larger LVC (Live, Virtual, Constructive) system could receive tracking data, use it to properly display 3D views of the virtual world, and then send it on to a VBS2 instance, allowing live and virtual trainees to interact with one another. When a client application is launched, it registers itself over the network with the MetaTracker server. In registering, the client sends the server data including a human-readable name (for verifying connectivity), and information about the physical area being tracked. This serves to reduce network traffic by only sending data where it is wanted or needed.

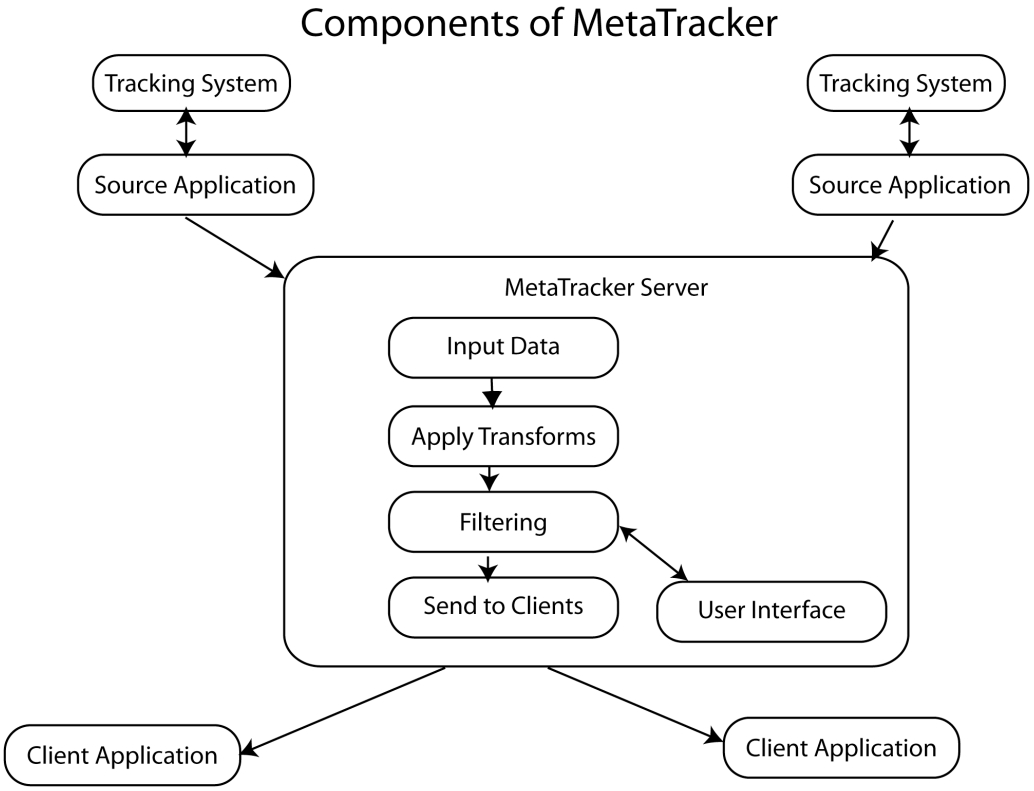

Figure 2: The components of MetaTracker, showing a pair of tracking systems whose data is accessed by multiple clients

Positional Errors in MetaTracker

In the development and testing of MetaTracker, one area that required a great deal of work was situations where two different systems were simultaneously tracking the same objects. This commonly happens in MRTS with multiple tracking systems as described earlier. Though carefully calibrated, the different tracking systems always had a small amount of difference in the reported position of an object. The initial behavior of the MetaTracker server was to immediately pass any tracking data received from any source application on to the client applications. In the case of two tracking systems reporting different values, the client would alternatively receive two different positions for the same object, resulting in the object appearing to "jitter" back and forth erratically between the two positions. This could create difficulties when the system is used for actual training. For example, if participants are attempting to fire simulated weapons at targets, their aim or the target position could be incorrect from one moment to the next. For a skill as precise as marksmanship, this could results in incorrect training and skills acquisition. There were also

concerns that if head-tracked positions experienced jitter, the resulting shifting of the 3D perspective could induce cybersickness in participants that stood in the area of overlapping coverage. For reasons such as these, jitter reduction became a high priority for MetaTracker development.

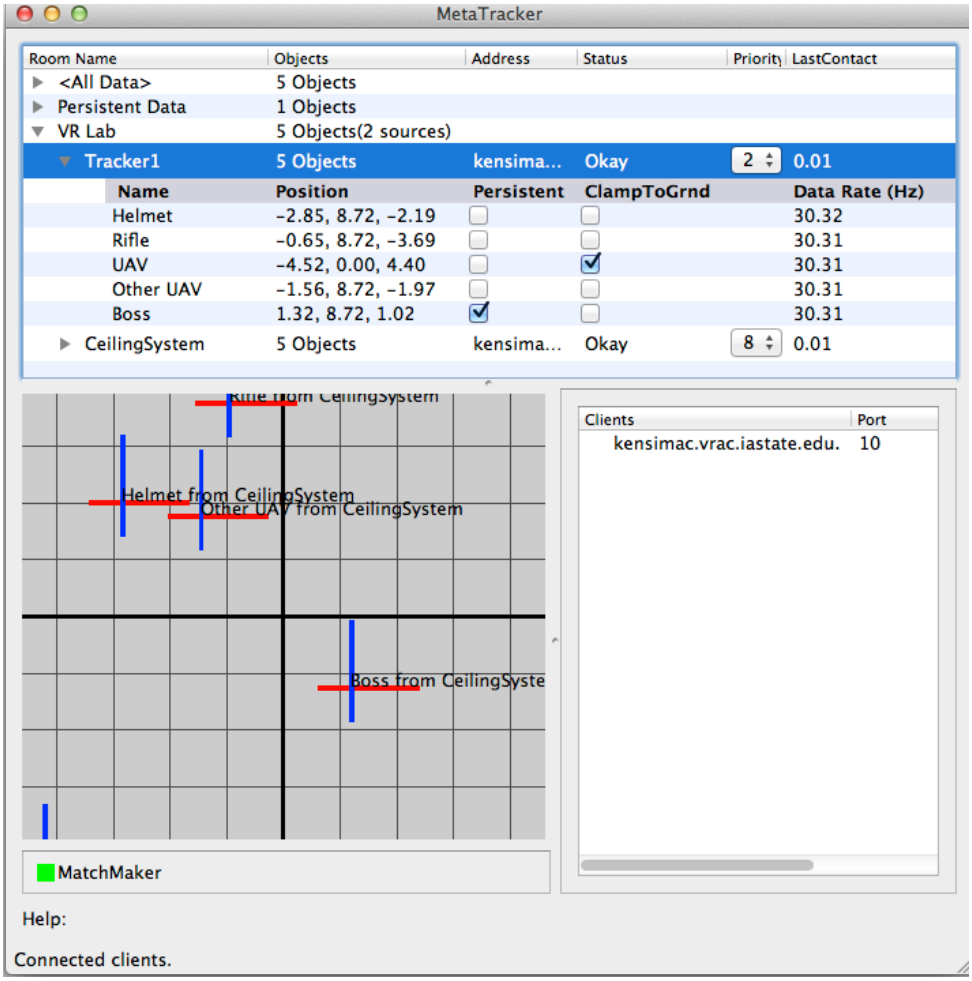

Figure 3: The user interface for the MetaTracker server, showing visual and textual data for tracked objects.

Initially, the only way to replicate this behavior, for development and testing, was to run another software application that would draw 3D models at the position of a tracked object, then walk across an area covered by two tracking systems. This method of testing was time-consuming and the data it produced was transient and couldn't be reliably replicated. Slightly moving the tracked object may reduce or eliminate the jittering error, thus making it difficult to troubleshoot and repair. At a minimum, two people were needed to work on this issue. One to move around the tracked area with a tracked object, while a second attempted to view the data, troubleshoot the problem, and alter the code in real-time. This proved to be a less-than-effective means to solve this problem. In addition, this only created the issue with a single tracked object. In a realistic training scenario, there would be multiple tracked objects moving in and out of these transition areas. This situation also had to be tested, but would require a number of humans to create this environment.

A second attempt to solve this problem was to record and graph the output of MetaTrackerTracker. This allowed an accurate visualization of the problem (again with human participation), but did not help with replication. While it offered additional information into what was causing the problem, it offered no way to test implemented solutions. In addition it was found that standing up and moving about the room to perform the test was disruptive to the thought process and further slowed development.

To replicate this problem, we created an application that simulated an entire tracked area. The application, called AreaViz, had several simulated tracking systems, each with its own artificial calibration error and a specified area it could "see." These simulated tracking systems each incorporated a MetaTracker source that used the same source code as those connected to real tracking systems. Like the real tracking systems, these sources periodically polled the data from the simulated tracking system and sent it to the MetaTracker server, running on a separate computer. To eliminate the need for people to walk around with tracked objects, simulated objects were instantiated that moved about the space, either in ellipses or straight lines, entering and leaving the tracked areas of different simulated tracking systems. A MetaTracker client built into the application received the data back from the server. All parts of the application were carefully timed using the computer's internal clock, so that simulated tracking data was sent at regular intervals. By using simulated objects and tracking systems, as well as receiving the processed data from the server, AreaViz was able to simulate and visualize all the data relevant to the multiple tracking system problem: actual object positions, object positions as seen by trackers, and object positions reported by MetaTracker. As illustrated in Figure 4, the green path shows the trajectory reported by the MetaTracker, while the purple line indicates the object's true (simulated) location. The oscillations shown in the green path are the result of jitter caused by calibration error of the two simulated tracking systems. This and other output from AreaViz greatly aided in recreating jitter and testing solutions for its reduction.

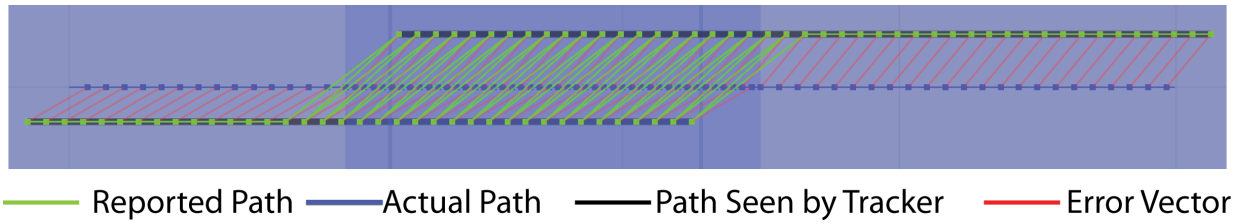

Figure 4. Visual output from AreaViz of an object moving across an area (darker blue region in the center) covered by two motion tracking systems of differing calibration.

In creating simulations of object motion and motion tracking systems, several simplifications were made. They were chosen in order to recreate the problem of jitter in a repeatable manner that was easy to analyze both visually and programmatically:

1. An object was visible to a tracking system if and only if it was within a rectangular region defined as that system's trackable area.

2. The measurement error of a particular tracking system was a constant positional offset.

3. If a tracking system could see an object, it sent data about its position at a constant rate to the MetaTracker server.

4. Tracked objects moved at constant velocity.

Using these rules for a simulated tracking systems, simulations were run that recreated the jitter effect. To provide a measure of jitter reduction, a method of quantifying jitter was devised, summing the frame-to-frame change in the error of MetaTracker's reported position. This measure was sensitive to jitter without being greatly affected by the error of the individual tracking systems. Visualizations and quantitative data created by the AreaViz application were used to test potential jitter solutions without having to use the immersive VR system. This is discussed more in the results section of this paper.

Latency Testing

When MetaTracker reached the testing phase, latency was one of the major potential shortcomings of the system. Latency, in this case, is defined as the elapsed time between an object moving and that motion being displayed in the virtual environment. Increased latency can result in a reduced sense of presence and immersion (Meehan, Razzaque, Whitton, & Brooks Jr, 2003) and reduced human performance (Juan & Perez, 2009). From a practical viewpoint,

when a trainee performs an action they expect immediate visual feedback as in the real-world (e.g., a weapon trigger is pulled and a bullet immediately is launched). Latency in a MRTS can frustrate trainees and even impede the training itself as tasks can't be completed in the same manner as in the field.

If it was to be a useful part of an immersive training system, MetaTracker had to add as little latency to the tracking data as possible. To test MetaTracker's latency under a variety of different loads, we developed an application called StressTester to measure the delay between the time data was sent to the MetaTracker server and the time it was received by clients. StressTester was a combination of a MetaTracker source and client. Like AreaViz, it had simulated objects moving around a tracked area, and it sent their positions to the MetaTracker server, but encoded into the y-coordinate of object positions was the send time of each data point. The flow of tracking data between the StresTester application and the MetaTracker server is shown in Figure 5. By comparing the time data was sent to the time the corresponding data was received, it was possible to accurately measure the latency of each data point for each simulated object. Command line arguments passed to StressTester told the application how many objects to simulate and how often to send. Using a shell script allowed for automated testing of a wide variety of loading conditions, with summarized results for each trial written to a single file. In order to more closely simulate actual use of MetaTracker, the StressTester application was run simultaneously in multiple instances from different computers during some trials.

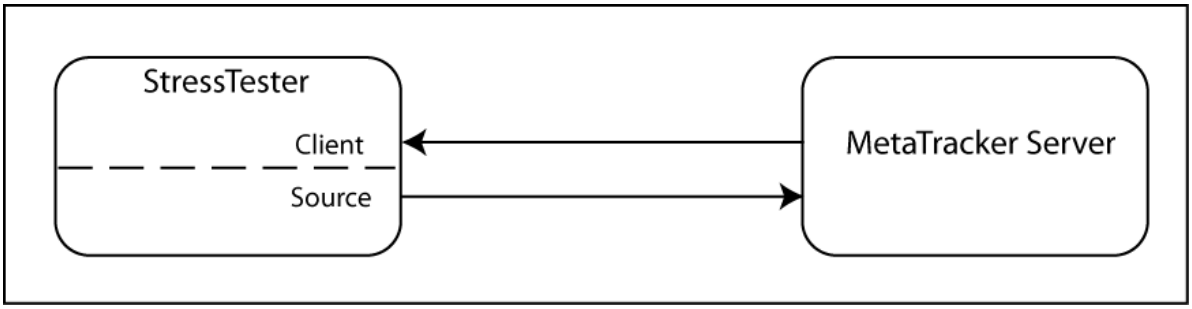

Figure 5: The flow of data when using StressTester for measuring latency.

RESULTS

Both testing applications provided valuable data for verifying and improving MetaTracker. AreaViz was used first to recreate and measure jitter and then to test possible solutions for reducing it. Each potential solution was tested, modified, and retested repeatedly. For latency and load testing, StressTester was controlled via a shell script in a variety of different object counts, data send rates, and in situations with and without additional (simulated) tracking systems interacting with the MetaTracker server.

Positional Error Testing

The use of simulated data generated by AreaViz greatly sped up the process of developing and testing MetaTracker. Changes to jitter reduction techniques could be tested in a matter of seconds, rather than a matter of minutes, and were accompanied by much more detailed data than could be obtained by watching a 3D shape bounce back and forth in a virtual environment. With the much faster code-to-results time allowed by the user of simulated data, data filtering in MetaTracker was developed and improved in much shorter time than it otherwise would have been, and without the need for additional personnel. This type of time and labor savings is important in creating software and hardware systems within budget and time constraints.

Load and Latency Testing

The StressTester application proved to be very useful in highlighting performance bottlenecks in MetaTracker. By configuring it to simulate dozens, or even hundreds of tracked objects, it was able to produce data at a much higher rate than could be practically achieved using real tracking systems. Used in combination with software time

profiling of the MetaTracker server, several areas were discovered where changes to the code resulted in better performance and lower system overhead. While this information was valuable, the more direct benefit of StressTester was its latency-measuring capabilities. StressTester was used to output files listing the send and receive times for every data point for every simulated object. Upon program completion, this data was automatically charted using the open source GNUPlot software, creating the graphs seen in Figure 6. Without automated tools using simulated tracking data, these graphs would have required a great deal of human labor to obtain, even for just a few objects. Some conditions, such as testing with 25 objects simultaneously, would have been impossible to test manually, as that many trackable objects were simply not available in the facility. The ability to test currently-impossible conditions is an important part of non-functional testing, and indicates future expandability of the system.

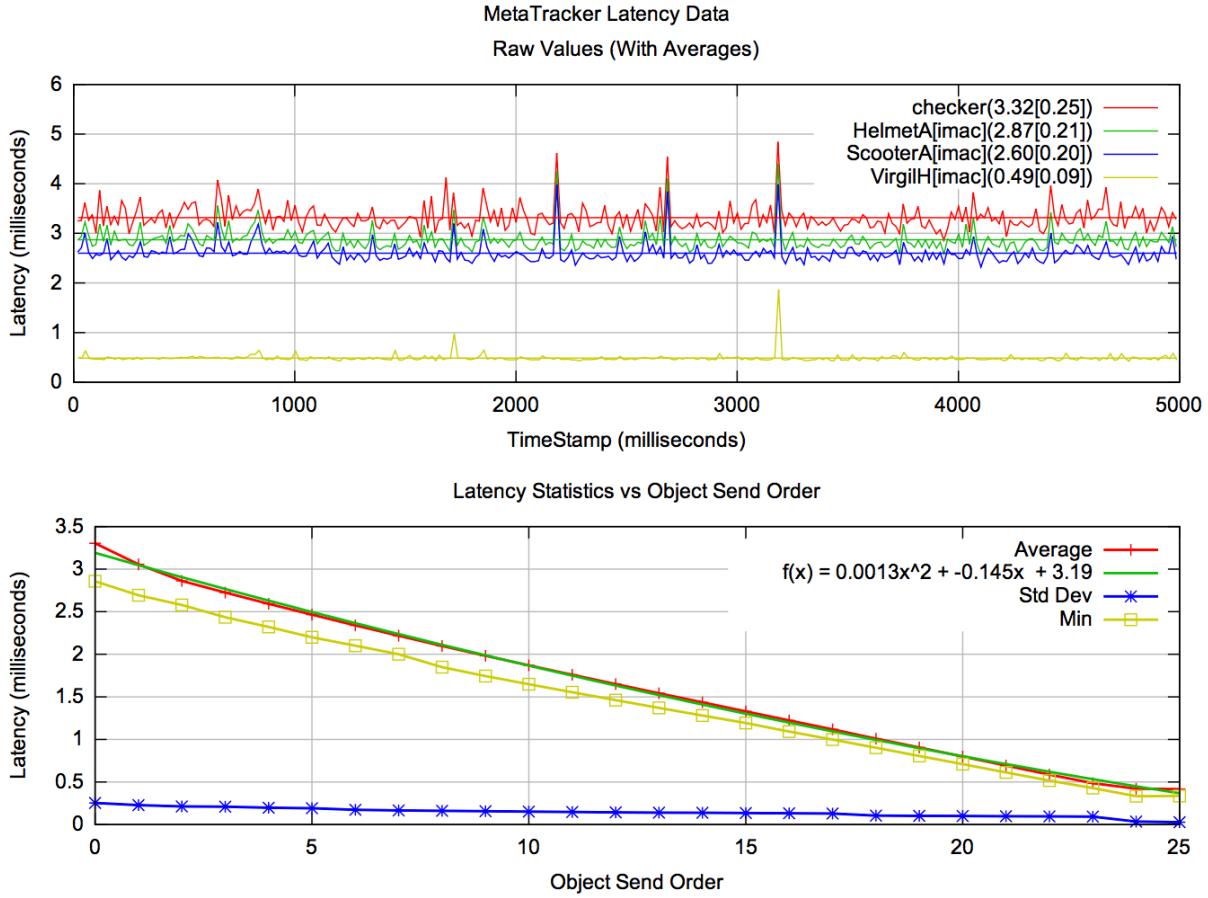

Figure 6. Round trip times for data sent to and received back from MetaTracker. For this particular trial, 25 objects were simulated in StressTester, sending data at a rate of 60 Hz over a UDP connection. In part (b), the relation between the order object data was sent in is shown compared to its latency time.

Bug Isolation

While the testing methods presented in this and the preceding section focus on using simulations to improve the server portion of MetaTracker, the AreaViz and StressTester applications both utilized the same code for sending tracking data and receiving it as the source and client parts of MetaTracker. This provided an opportunity to view the behavior of those parts under high load conditions, which accelerated the appearance of a few harder-to-replicate bugs, such as slow memory leaks and race conditions. Under normal use conditions, these "Heisenbugs" tended to vanish when efforts were made to track them down (catb.org, 2014). Stress testing MetaTracker caused them to manifest themselves within seconds instead of minutes or even hours, making them much easier to find and fix.

CONCLUSIONS AND DISCUSSION

Using simulated inputs for MetaTracker had a tremendous impact on its development and testing. Jitter was greatly reduced, and in some cases eliminated entirely. The latency testing results in Figure 6 show that the measured latency values were low compared to overall VR system latency of 50-60ms found by (He, Liu, Pape, Dawe, & Sandin, 2000). The precisely repeatable conditions created by AreaViz took a great deal of guesswork and uncertainty out of the development of MetaTracker's data filtering. Lastly, the aforementioned Heisenbugs that stress testing helped to eliminate helped to elevate MetaTracker from a useful experiment to deployable software.

Developing simulated inputs and automated testing methods for MetaTracker was a relatively straightforward task, and the benefits realized greatly outweighed the effort required. While the tools developed in this paper are specific to MetaTracker, the ideas behind them—generation of simulated data and automated use of this data for testing and evaluating—are not. Applying these principles to the development testing of other military training systems potentially achieve similar benefits. The current widespread use of simulations means that many systems can be easily tested with simulated data, accelerating development and refinement.

REFERENCES

Apple. (2014) Retrieved 5-8, 2014, from https://developer.apple.com/library/mac/documentation/AnalysisTools/Reference/Instruments_User_Reference/UserInterfaceInstrument/UserInterfaceInstrument.html

BohemiaInteractive. (2013). VBS2 | Bohemia Interactive Simulations, from http://products.bisimulations.com/products/vbs2/overview

catb.org. (2014). Heisenbug, from http://www.catb.org/jargon/html/H/heisenbug.html

Chisnell, D. (2009). Usability Testing Demystified Retrieved 5/19, 2014, from http://alistapart.com/article/usability-testing-demystified

CRYTEK. (2013). CryENGINE Retrieved May 28, 2013, from http://www.crytek.com/cryengine

He, D., Liu, F., Pape, D., Dawe, G., & Sandin, D. (2000). *Video-based measurement of system latency.* Paper presented at the International Immersive Projection Technology Workshop.

Juan, M. C., & Perez, D. (2009). Comparison of the Levels of Presence and Anxiety in an Acrophobic Environment Viewed via HMD or CAVE. *Presence-Teleoperators and Virtual Environments, 18*(3), 232-248. doi: 10.1162/pres.18.3.232

Lutz, R., Hanrahan, T., Schneider, D., Edwards, M., Graeff, R., & Gould, N. (2013). *Advanced Modeling and Simulation Techniques for Evaluating System-of-Systems Performance.* Paper presented at the The Interservice/Industry Training, Simulation & Education Conference (I/ITSEC).

MacLeod, M. J. (2013, Feb 08, 2013). Virtual Convoy Simulator Maximizes Training Time. *Military.com*.

Martin, G. (2013). Laser Shot bringing space-age firearm training to Africa Retrieved March 4, 2013, from http://www.defenceweb.co.za/index.php?option=com_content&task=view&id=29079&Itemid=109

Meehan, M., Razzaque, S., Whitton, M. C., & Brooks Jr, F. P. (2003). *Effect of latency on presence in stressful virtual environments.* Paper presented at the Virtual Reality, 2003. Proceedings. IEEE.

Pal, K. (2014). Understanding Non-Functional Testing Retrieved May 7, 2014, from http://mrbool.com/understanding-non-functional-testing/29689

Pan, J. (1999). Software Testing Retrieved May 8, 2014, from http://users.ece.cmu.edu/~koopman/des_s99/sw_testing/

Pollock, B., Winer, E., Gilbert, S., & de La Cruz, J. (2012). *LVC interaction within a mixed-reality training system.* Paper presented at the IS&T/SPIE Electronic Imaging.

Zhou, S., Chen, D., Cai, W., Luo, L., Low, M. Y. H., Tian, F., . . . Hamilton, B. D. (2010). Crowd modeling and simulation technologies. *ACM Transactions on Modeling and Computer Simulation (TOMACS), 20*(4), 20.