

Technology Challenges of Distributed, Multiplayer Virtual Environments

Jon Watte
Chief Technology Officer
Forterra Systems Inc
jwatte@forterrainc.com

October 2006

*Material presented in this document is covered under
US Patents 6,545,682, 6,628,287, and 6,784,901.*

1 Introduction

Popular on-line computer games have blazed a trail in fielding capabilities that are interactive, collaborative, immersive and generally entertaining. Games like Counter-Strike™¹, Runescape®, World of Warcraft® and The Sims Online™ show that it's possible for thousands of people to interact in a shared virtual space. The ability to capitalize on the underlying technology and adapt online games into accessible, effective online training, collaboration and experimentation systems is an obvious next step. Lack of instructors, distributed participants, or remote locations no longer have to prevent effective delivery of training or collaboration when all that's needed is a modern personal computer and an internet connection.

However, supporting large-scale persistent 3D virtual worlds that are accessible by multiple distributed users using standard networks, bandwidth, and PCs presents a series of complex technical challenges that include:

- Network Topology
- Load Distribution
- Responsive Visualization and Interaction
- Physics and Latency
- Long Distance Object Interaction
- Interaction with Legacy Simulations
- Data Capture and Review

This paper describes each challenge, reviews various solutions, and describes the implementation developed by Forterra Systems in its On-Line Interactive Virtual Environment (OLIVE) Platform.

Founded in 1998, Forterra Systems, Inc. has invested over \$50M into the development of 3D large scale persistent virtual world technology that allows military, healthcare, government, education and enterprise organizations to train, plan, and collaborate in ways that have not previously been possible. Forterra Systems' technology allows users to create, operate, and modify persistent, distributed, three-dimensional virtual worlds that can be populated in real-time by many thousands of simultaneous users — each represented as 3D entities — over a LAN, WAN, or Internet.

Today, the OLIVE Platform supports a variety of organizations for a wide range of applications including FDNY, the SUMMIT Medical Center at Stanford, the Wallenberg Foundation and the US Army's Research, Development and Engineering Command (RDECOM). Applications already developed with OLIVE range from an Incident Commander application that is able to properly replicate the way that a fire captain interacts with different fire crews in a high-rise fire to the Asymmetric Warfare-Virtual Training Technology (AW-VTT), which supports joint, interagency, and multinational operations in asymmetric and unconventional warfare, including antiterrorism, force-protection and security and stability operations (SASO).

The same 3D, large-scale persistent world technology is behind the consumer virtual world There (www.there.com) that was successfully launched in October 2003, and the new on-line service offered by MTV, known as Virtual Laguna Beach (www.vlb.mtv.com). These services provide a seamless environment that supports thousands of concurrent users accessing the environment from around the world.

¹ Counter-Strike is a trademark of Valve, Inc.; Runescape is a registered trademark of Jagex, Ltd.; World of Warcraft is a registered trademark of Blizzard, Inc.; The Sims Online is a trademark of Electronic Arts.

2 Platform Challenges

This section reviews the most significant networking and simulation challenges that must be addressed when building a 3D large-scale persistent world technology platform. For each challenge, we review various solutions, and describe the implementation chosen for OLIVE.

2.1 Network Topology

Due to the inherent complexity of natural phenomenon in the real world, large-scale virtual worlds require significant computing power to simulate. Assuming that each user will only have a single client computer, a powerful server is typically required to host the world assuming a reasonable fidelity world simulation is required. Some virtual environments use the client machines to do all or part of the simulation work thus minimizing or eliminating the need for a central server. While this peer-to-peer configuration appears advantageous from a hardware requirement standpoint, this approach has some serious limitations.

- Network bandwidth and latency issues are likely to limit the simulation to co-located clients.
- The slowest client machine limits the simulation experience for all users.
- It is possible for a client to be tampered with and broadcast inaccurate information.

The last point is a major concern for game developers and should be considered by anyone building virtual platforms if they intend to leverage their technology for mission critical applications beyond games.

The alternative to a peer-to-peer configuration is a classic client-server topology. In this topology, there is still a question of whether to use a thick or thin client approach. Thin clients have been used by a number of companies to minimize the client hardware requirements and achieve the maximum level of client machine compatibility. Unfortunately, thin client solutions require frequent object state updates from the server and actions taken by the user are delayed by at least one network roundtrip to the server. Both of these issues dictate high quality network connections and ultimately preclude modem and very long-distance broadband connections, for example, from Asia to the continental United States.

Forterra Systems specifically designed the OLIVE platform to allow low-bandwidth client connections and prevent pitfalls associated with a peer to peer network. More information on how

this is accomplished is provided in Sections 2.3 and 2.4.

2.2 Load Distribution

In designing a persistent virtual world, one of the first challenges is how to distribute objects across multiple servers. Depending on scale, virtual environments may be required to simulate anywhere from many hundreds to hundreds of thousands of objects. Hosting all objects on a single server may work for small-scale virtual environments, but will most certainly fail as the environment becomes more complex. A distribution methodology must be adopted that easily scales, minimizes cost of operation, and maintains runtime performance independent of simulation size.

One approach is to assign objects to a particular server based on object type. For example, one might deploy three servers to handle a simulation containing helicopters, cars, and pedestrians. While better than locating all objects on a single server, this solution has significant downsides because it depends on predefining where object types are simulated and fails if the number of any one object type exceeds the capacity of its respective server.

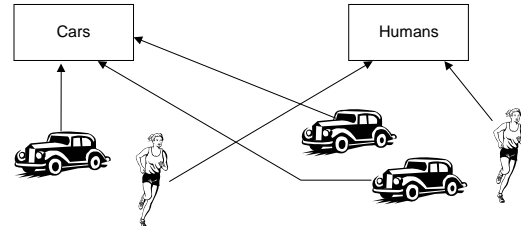


Figure 1 One Server Per Object Type

Some scalability issues in this approach could be addressed by having multiple servers per object type. This works well when objects do not interact frequently or necessitate low latency connections. For tightly coupled, highly interactive, modern simulations, however, this approach becomes cumbersome.

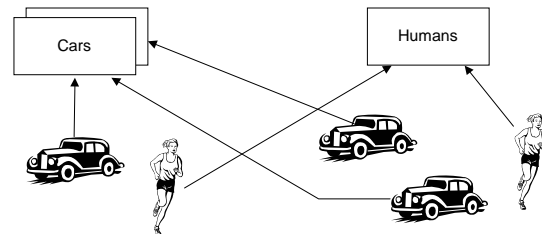


Figure 2 Many Servers Per Object Type

A third approach is to split the simulated world into multiple sectors, and assign a single server

to each sector of the world. Any object that is located in a specific sector is simulated on the server for that sector. When the object moves into a new sector, responsibility for the object is handed off to the new sector server. Scalability is much better than with the previous approach because object interaction is almost always dependent on object proximity and the number of objects that interact across different servers is small. Forterra Systems believes this approach leads to the highest system efficiency, and thus to the lowest cost of operation. OLIVE aggressively makes use of locality of simulation to allow objects to interact efficiently, while still allowing objects on different servers to interact with low latency.

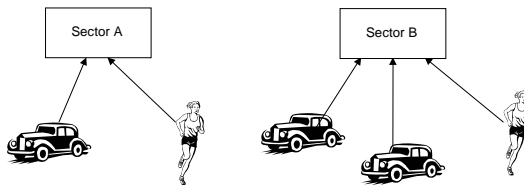


Figure 3 One Server Per Geographic Sector

OLIVE splits the world into a number of geographic sectors whose size and shape vary depending on the virtual environment. The granularity of a sector may be as small as 50 meters or extend out for thousands of kilometers and, in fact, cover the entire world. OLIVE sectors are not static and may be adjusted at runtime to maintain a consistent load balance amongst the servers. This is important for simulations where large numbers of objects can converge on a single location whose associated server might otherwise become overloaded.

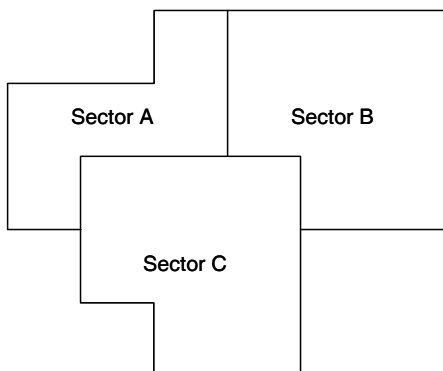


Figure 4 Irregular Shapes of OLIVE Sectors

Objects in a simulation may interact even if different servers host them. This is especially important for interactions that take place over a great distance, like a weapon firing at a target

many kilometers away. To ensure that all relevant surrounding object data is available to all simulated objects, each simulated object is assigned a radius of interest, and the system ensures that any simulated object can see a fully physically accurate model of any other object within its radius of interest. This is a stronger guarantee than legacy simulations, which typically only show a dumbed-down view of the simulation (such as a dead reckoned entity) to other simulated objects.

All components of OLIVE are designed to scale horizontally like this; from simulation servers through databases to voice-over-IP communications — so if you need a larger scale capability, simply adding more hardware will allow you to scale to meet the requirement.

2.3 Responsive Visualization and Interaction

Real-time interaction is a hallmark of 3D virtual environments, but maintaining the level of responsiveness required to create *realistic human interaction* is challenging, especially as the number of simulated objects increases. To mimic the real world, all participants in the virtual world must be able to affect the virtual environment and receive immediate feedback from their actions.

The result is that to maintain consistency across a distributed 3D virtual environment, a significant amount of data must flow between clients and servers, and data latency becomes a major obstacle to delivering a smooth, natural experience for all participants. An unacceptable level of “jitter” would be introduced if every computer had to pause for the entirety of some specific message to arrive. Further, if the simulation involves participants connecting through dial-up modems, the available bandwidth is very low. Taken together, these requirements pose a significant challenge to the development of robust, scalable, interactive distributed simulations.

One approach to real-time interaction is to co-locate all client and server machines while using low-latency, high-bandwidth network connections. This way, the entire simulation state can be sent to each client at a relatively high frequency and the client can use dead reckoning to estimate object state between updates. This approach is used by a number of consumer game engines, but has several weaknesses:

- Physical proximity requirements between clients and servers present significant logistical problems.
- Sending complete simulation state across the network makes inefficient use of available networking bandwidth.
- Dead reckoning may cause participants to see things that didn't actually happen.
- In the end, relying on "faster networks" puts a hard cap on the ability to scale to really large implementations.

Figure 5 depicts a situation where dead reckoning causes someone to see something that did not happen. A car that drives towards the edge of a river bank, but changes direction at the last moment may appear to have driven into the water until the next state update arrives and the visualization is adjusted.

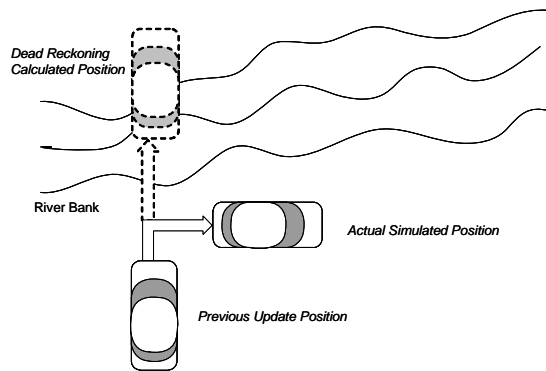


Figure 5 Problematic Display Resulting From Dead Reckoning

Forterra Systems believes that a 3D virtual world platform must be designed from the ground up to tolerate latency in networking and to make extremely efficient use of available networking infrastructure. The platform should only render events that have actually transpired, thus guaranteeing that the position and behavior of objects is always correct according to the physical rules of the simulation.

OLIVE implements the specialized TVP protocol to guarantee responsive visualization and interaction. Developed by Forterra Systems, TVP is a state-reliable protocol that ensures that object state undergoing co-simulation on multiple hosts will eventually reach consistency, even under adverse network conditions. TVP was developed to address two design principles:

- Overall object state is more important than the individual network messages.
- With identical inputs, an object can be identically co-simulated on multiple machines.

During a simulation, individual network messages do not matter as much as the overall state of an object. If a specific message is lost, re-sending that same message is unimportant because the re-sent message will arrive too late to be relevant. Communications should be based on replicating simulation state to all involved participants. If one state update is missed, the next state update should be as fresh as possible when it arrives, not just a verbatim copy of the previous update.

With identical inputs, running in an identical environment, an object can be identically co-simulated on multiple machines. Therefore, only portions of the object state that cannot be derived from other state information need be transmitted. By co-simulating objects on client machines and in some cases multiple servers, the amount of data transmitted is minimized and primarily reduced to unpredictable user input to the simulation. Regardless of the number of machines co-simulating an object, there is always one authoritative server.

TVP was designed to be state-reliable and to support the notion of user input-driven co-simulation to reduce bandwidth needs while preserving fidelity and responsiveness to change. Further, since OLIVE was designed for non-homogenous hardware and operating systems environment, the platform automatically handles threats to simulation consistency such as variations in floating-point library code and FPU (floating point unit) control words on different platforms.

Forterra's TVP protocol classifies data it exchanges into three categories which are described in Table 1.

TVP Data Class	Description
UI	User Input that changes the behavior of objects and that cannot be predicted or co-simulated by nature.
Commands	Messages indicating specific decisions made by the authoritative copy of an object, in cases where co-simulation would be inaccurate.
State	The actual state of an object, sufficient to re-animate it on another simulation host or client.

Table 1. TVP Data Classes

TVP is layered on top of UDP, the industry-standard IP protocol used for real-time traffic such as voice communication or video streaming. UDP is a lossy protocol and does not

guarantee that a specific message will make it across the network; although, in practice, almost all messages make it to their destination. In return for accepting this potential for loss, UDP affords the system the lowest possible transmission latency and network overhead. For efficiency reasons, many UI, State and Command messages are combined into a single UDP message by TVP — therefore minimizing framing and protocol overhead.

The lossiness of UDP does pose a challenge when a message containing Commands or UI is lost and the recipient can no longer correctly co-evolve the simulation of the intended recipient object. TVP cooperates with Diphy (Section 2.4) to solve this problem by continually calculating and exchanging checksums of object state. When the checksums differ, the authoritative state on the server will be re-supplied to the client as a correction in state, and the simulation can proceed in sync again. The checksums, which are about 1% of the size of typical state updates, are calculated and exchanged on a robust schedule to minimize the time during which an object may be out-of-sync after network packet loss. While a false positive match may happen one or a few times, they are guaranteed to mismatch with enough iterations of checksums for any given object with an actual discrepancy.

To further improve scalability, Forterra has developed high-performance router software for TVP that can off-load most of the visible-object determination and interest management for clients and other simulations from the simulation servers and provide a practically unlimited number of participants a view into the simulation. Clients only receive information about objects that meet certain thresholds. For example, the client may want to see avatars (a simulated virtual human controlled by the user) within a half-kilometer, buildings and vegetation within two kilometers, and basic terrain data within ten kilometers. The client machine sets thresholds based on machine performance and the nature of the virtual environment application.

Participants in virtual world applications often expect to see certain visual effects that do not significantly affect the outcome of the simulation. Visuals like smoke from a discharged weapon, while important to make visible, do not need to render identically on every client machine. OLIVE accounts for this by only sharing primary state information with the clients (e.g. a weapon

was fired or a vehicle takes damage) and then allowing the client to determine how detailed to make these effects, or whether to render them at all. This minimizes server load and network bandwidth consumption.

2.4 Physics and Latency

The challenge of supporting real-time interaction within a 3D virtual environment is greatly simplified when all objects are simulated on the local machine rather than having to forward interactions between machines. However, in a distributed simulation one must account for the fact that the authoritative copy of each object may not reside on the same machine. This is particularly true when that machine is a client accepting user input.

If it is not possible to have the authoritative copy of all objects reside on one machine, one can at least have co-simulated proxies to the authoritative object. If the authoritative home of an object is on another machine, that proxy copy needs to be in sync with the authoritative home of the object. A good solution is to make sure that when authoritative object A sees proxy instance B, the master object B also sees a proxy for A. Further, all objects that can affect object A are visible to A, and A is visible to all such objects. Finally, the state of all these instances needs to be the same across all simulation hosts for a specific simulation point in time. This is in contrast to legacy simulation technologies where object A will only see a rough approximation of object B, which may lead it to draw the wrong conclusions about that object.

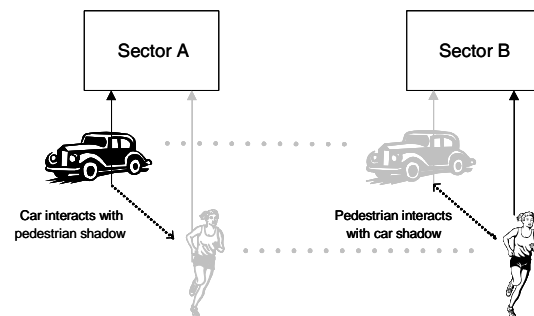


Figure 6 Shadow Objects Provide Low-Latency Interaction Ability

When some of the simulation happens remotely — e.g., a remote user connected through a modem issues a command to fire a weapon — the latencies involved in network transmission make the aforementioned approach difficult.

Either the simulation has to wait for each object to complete each simulation step so that it knows it has enough information to continue the next step, or the simulation will have to delay the issuance of the fire-weapon command until the command has been transmitted to all participants in the simulation. Unmitigated, neither of these solutions will achieve acceptable interactivity.

Forterra has built a real-time, interactive, distributed simulation technology called *Diphy*, which ensures that events always happen on a consistent time step for all proxy and authoritative object instances, while allowing remote users immediate feedback on their actions. Furthermore, the simulation is synchronized across an entire cluster of servers 30 times a second, while tolerating remote clients with a one-way transmission latency of up to 600 milliseconds (twice the latency of a typical cross-continent dial-up modem connection).

At its core, *Diphy* is a time-step based simulation with support for high-order integration, collision testing and response, force accumulation and reaction to user input. Unlike competing technology, however, *Diphy* collaborates with TVP to enable “time dilation” which brings low-latency, high-performance responsive simulation to all participants, even if they are connected through a dial-up modem.

In most distributed simulations, time is divided into successive discrete periods known as time steps. The state of the simulated world is advanced from one such time step to the next by applying rules of the simulation, which usually involve Newtonian physics of gravity, opposing forces of contact, force applied by engines, etc. The size of the time step and the algorithms for advancing the simulation are chosen to maximize the accuracy of the simulation, while keeping simulation costs within a useful budget (in machine resources and, ultimately, monetary cost).

The time steps that advance the world are called “virtual time,” as opposed to the time you can measure with a regular clock which is known as “wall-clock time”. In a real-time simulation, virtual time and wall-clock time advance at the same rate. The virtual time kept on the simulation servers is known as “global virtual time”.

In a distributed simulation, the additional challenge of transmission latency means that a command issued at a certain virtual time will arrive at other hosts (such as the authoritative

simulation servers) at some later virtual time. In a co-simulation system where the simulated objects have simulated shadows on the client machines to aid display of the simulation, one possible solution is to show all objects at a time that’s one maximum round-trip time delayed from the server virtual time. However, this leads to non-responsive interaction for the user; if the round-trip time is 200 milliseconds, there will be a 200-millisecond lag between the user issuing a command and the effects of that command being displayed to the user.

To understand how *Diphy* solves this problem, some terms need explanation: *Diphy* classifies objects in the classes of “actors” and “obstacles.” An avatar or other object being interactively affected by a user is said to be an “actor” whereas an object that does not receive direct user input is said to be an “obstacle.” The local copy of an object being affected by the user is said to be the “pilot” for that object for the simulation. This does not mean that the pilot is authoritative for the simulation; it means that the pilot is the object that is allowed to send direct input relating to an object to the server for that object.

Diphy solves the transmission latency/responsiveness problem by allowing different objects to be simulated and shown at different times on different machines. The main user avatar and objects directly controlled by the user are displayed ahead of global virtual time by an amount that equals to the transmission delay between the active user and the server. This allows commands issued by the user, to control the avatar and surrounding objects, to have immediate effect on the client machine, while being sent to the server to arrive in time for the server to apply them on the same time-step for the same object. Commands are also forwarded to other clients so they can co-simulate the same changes on their shadow copies of the avatar and affected objects. It follows that client machines display other avatars, and objects that other avatars affect, at a time step that is behind the global virtual time by an amount that equals to the transmission delay from the server to that user. This way, the input from one user will be available to the co-simulation on the other user’s machine in time to display a physically correct simulation of the first user’s avatar.

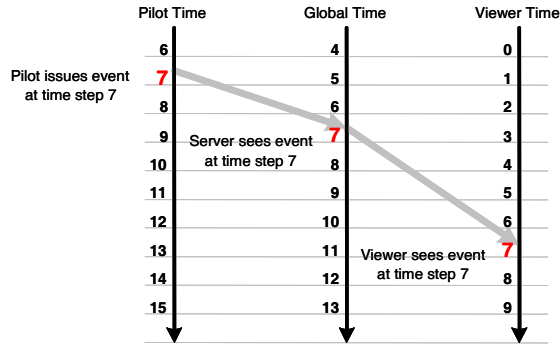


Figure 7 Running Objects Time Shifted Ensures Consistent Views

The end result is that all objects run at the same global virtual time on the authoritative server, but are displayed and co-simulated at different virtual times for a given wall-clock time on viewing clients. The delay of objects and avatars not under the client's immediate control is proportional to the message propagation delay from the server to the client. In the implementation of each object, a log is kept of object state and received interactions so that a state update received for a time step in the past can re-play other received events and still end up with a consistent object state.

The notion of "objects under the client's control" needs to be adjusted for a smooth experience. Think, for example, of a simulated pendulum that swings back and forth as the pilot avatar walks up to it. Initially, the object is not under control of the pilot and thus its display is time delayed. However, as the pilot gets close to the object, interaction becomes more likely, and the pendulum should be brought into the pilot's time frame of reference, and thus run forward extrapolated (based on available events) on the pilot's machine. OLIVE implements an algorithm that uses distance between objects and the pilot to continuously adjust the trade-off between relying only on received data and providing interactivity for nearby objects. In practice, this leads to a simulation with a high degree of realism, interactivity, security and integrity. The end result will be that as the pilot walks closer to the pendulum, and it is brought further forward into time, a kind of Doppler shift effect makes the pendulum swing faster while being approached. This Doppler shift is usually not noticeable and vastly preferable to alternatives such as "snapping" the time frame of the object forward as the avatar actually starts interacting with the object. Interaction between objects at a long distance can still be supported, as shown later.

Having different objects display in different frames of time adds a requirement for some degree of finesse in how interactions between separate actions are displayed and resolved on each client, as well as on the server. Actors not part of the direct interaction will always see exactly what happens on the server, because each interacting actor is equally time delayed on their display. Actors taking action, and actors being acted upon, have a choice in how responsive the action is, versus how likely it is that one of the actors (actor or actee) may see a display that reflects a sudden action in the near past for that actor. This tradeoff is made depending on the application requirements.

2.5 Long Distance Object Interaction

One challenge in traditional distributed simulation systems is efficiently simulating entities that cover a very large area. OLIVE solves this by combining the seamless transition of an object between simulation servers with the ability of an object to have a very large visibility radius. The guarantee is that an object will see instances of other objects with overlapping visibility radii. This is implemented by making sure that an object will be co-simulated on any simulation host that has map area that overlaps the visibility radius of every other object.

Using the example sector map from Figure 4, adding some units of measurement, and an assumed visibility radius of a long-range observation system of 350 km, this concept is illustrated in Figure 8, where the simulated system A will see simulated object B within its range, even though it is not on the same simulation server; meanwhile, it will not necessarily see object C, even though it is on the same server.

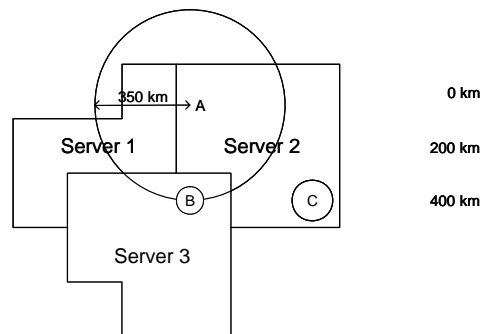


Figure 8 Visibility Determined by Range Across Simulation Hosts

A higher fidelity simulation would also include simulated phenomena like atmospheric scattering

or terrain blockage, some of which are supported directly by OLIVE and others which may be developed and controlled entirely by the customer.

This same approach can be used for objects that have a physical effect far away, such as missile systems or rail guns in defense applications, or large contamination plumes in a simulated first-responder exercise. If clients of limited capacity are employed to visualize or interact with the simulation, the number of objects with large interaction radius needs to be limited in order not to overwhelm the clients with co-simulation processing.

When an interactive object affects an object that is far away, that object will not be forward extrapolated on the pilot object's client machine. In this case, the effects of the interaction can be delayed by one round-trip latency (used for example for traveling artillery or rail-gun ordinance) or a correction into the past has to be accepted for the remote object being affected. For large, massive objects like artillery, ships, trains, plumes of dust etc, one round-trip latency is quite acceptable and often the best solution, but the specific solution chosen needs to be carefully selected with attention to the subject matter being simulated.

3 Interoperability

Today's training and experimentation systems rarely have the luxury of being stand-alone systems. More often than not, multiplayer systems must interface with a variety of real-time systems and legacy technologies. Support for government interoperability standards must be provided where interoperability with existing programs can be provided to leverage previous investments. For the systems to interoperate, the ability to match internal data structures and formats and timing must be provided to synchronize disparate systems. Levels of

simulation scalability must be sufficient to compensate for the increased loads. OLIVE provides these features, along with interest management and user interactivity for simulations that previously had not provided these features.

3.1 External Simulation Interoperability

In some cases, a 3D large-scale persistent virtual world can be a platform for integrating disparate data streams and models into one cohesive simulation environment. The virtual world allows one to naturally visualize external data and intuitively manipulate objects in ways simply not possible with conventional applications. Forterra's OLIVE platform is designed to ease data integration and assist developers in building powerful integrated applications as described below. The implementation uses a Gateway that allows external data sources such as semi-automated forces (SAF), global positioning sensor (GPS) data, medical models and live data feeds to pass into the OLIVE core. Figure 9. The OLIVE Gateway converts state information between the external format and that defined by TVP. Because TVP handles time management and all communication to the OLIVE simulation servers and other clients, the adapter is easy to construct.

Once information is passed to the OLIVE simulation servers via TVP, a Proxy Object can be used to synthesize intermediate state information that may be required by the simulation but is not provided by the external data source. For example, a GPS data stream from an air vehicle may provide position information, but not attitude data which is required for realistic rendering. The Proxy Object would compute this information and pass it, along with the original position data, to the rest of the simulation.

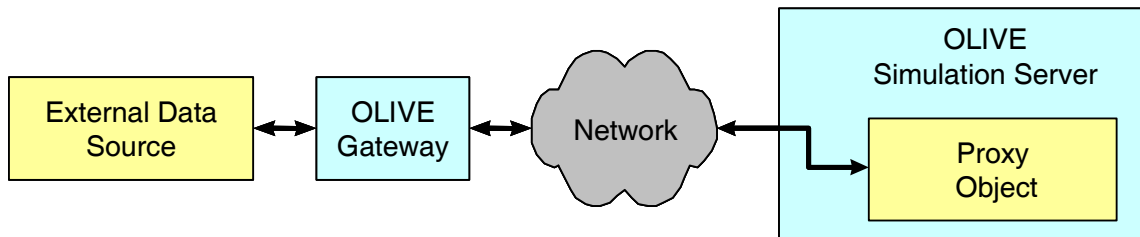


Figure 9 External Data Integration

The Proxy Object is implemented to reside on the simulation server, thereby guaranteeing low latency, high performance access to relevant

simulation data. Depending on the object being integrated, the Proxy Object may become fairly complex; however, the OLIVE platform provides

complete access to information about the object's surroundings, state history, etc. Further, all rendering and communication with the simulation clients is handled by the platform.

Positioning the OLIVE Gateway at the location of the external simulation or data source makes it convenient to integrate remote simulations using the OLIVE platform.

3.2 Content Interoperability

Key to interoperability of multiplayer systems and legacy simulations is also content or database interoperability. OLIVE has tools for exporting terrain and cultural data found within the simulation to correlated, industry-standard geometry formats for further import into legacy simulation systems. For example, a correlated database of central Baghdad can be made available in the internal OLIVE, SEDRIS and CTDB formats as required by U.S. Federal Government programs.

4 Data Collection and Session Review

OLIVE has a broad application programmer's interface (API) which provides access to the internal data model as well as properly authorizing external third parties and allowing data to be input to OLIVE. This interface is used for applications ranging from model-based analysis of trainee performance to a full-featured virtual possession auction and trading system.

4.1 Interest Management

Part of the OLIVE platform is a sophisticated interest management and data routing system, embodied by the OLIVE Visualization Server. Because data about all events in the OLIVE simulation passes through these servers, they are the ideal place to record and log all events about the simulation for later playback and analysis. Because the OLIVE client gets its entire view of the simulation through a network connection, the visualization server can re-create a data stream for a client from any point of view.

Forterra leverages this capability to provide an unprecedented level of session review presentation, where a presenter, distributed anywhere in the network, can fast forward and rewind the session showing the results to local or distributed participants from various points of view. Because data is centralized on the server side, but can be easily viewed by the clients,

centralized management of saved session recordings is easy to facilitate.

Another challenge for distributed multiplayer systems is data logging. The API to the OLIVE platform allows access to a built-in data logger, which extracts data about events in the simulation that are suitable for translation and loading into a data warehouse based on any of a variety of commercial database technologies (Oracle, MySQL, etc). Data warehousing logs are generated at very high resolution, with available fields including entity position, orientation and look direction with up to second resolution, entity fire interactions, trainee inventory operations, attendance times and duration, and many others.

5 Summary

Building a robust, scalable, secure and high-performance virtual world simulation system is very challenging and requires careful attention to detail. At the core, networking, simulation and interest management must cooperate to take best advantage of available bandwidth, and provide a consistent view of the simulation to all participants.

The Forterra OLIVE platform is representative of a system capable of supporting these requirements by integrating the Diphy distributed physics system with the TVP simulation view protocol and the OLIVE cluster topology to provide best-of-breed massively multiplayer simulation capabilities.

Being a next-generation system, OLIVE can also integrate and enhance external simulation data to preserve existing investment in legacy simulations.

6 About the Author

Jon Watte is Chief Technology Officer at Forterra Systems. He is also respected in the game development community with an active role in driving game development forward as a science.

Prior to joining Forterra Systems, Jon was Engineering Director at Be Incorporated with overall responsibility for architecture and delivery of the BeOS media, data translation, and selected kernel subsystems. Jon came to Be from Metrowerks Incorporated, where he worked on the CodeWarrior compiler tool chain for MacOS, BeOS and other targets. Prior to that, Jon worked on financial systems and workflow applications for pre-press and publishing in Stockholm, Sweden. Jon attended the Royal Institute of Technology in Stockholm.