Dynamic Synthetic Environments Through Run-Time Modification of Source Data

Kenny J. Hebert, Dale Sexton
Presagis
Richardson, TX
Kenny.Hebert@Presagis.com, Dale.Sexton@Presagis.com

ABSTRACT

One of the key components to achieving a high level of realism in today's training simulations is the ability to dynamically interact and manipulate the synthetic environment utilized by the simulation. Whether the interaction is the result of naturally occurring environmental effects, such as flooding and erosion, or simulation driven events, such as bomb craters, dynamically manipulating the synthetic environment provides critical feedback to the warfighter that is representative of real world engagements. In recent years, many research efforts have been undertaken to support the correlated representation of dynamic terrain in a federated simulation environment. However, much of those research efforts have focused entirely on the dynamic manipulation of the underlying terrain polygons and associated visual representation without considering the dynamic modification of the original source data or other derived datasets that are utilized by client devices participating in the training event. In today's Modeling and Simulation (M&S) environment, there are a wide number of applications and client devices that utilize native GIS source data at run-time and require more than just updated terrain polygons and visual images. In order to support dynamic modifications to the synthetic environment applicable to a full range of M&S applications, a comprehensive approach must be taken to ensure that multiple data types and formats can be dynamically modified and updated at run-time. In this paper we will provide an overview of the current approaches for achieving dynamic terrain and present a proposed method for the dynamic run-time modification of synthetic terrain data to support correlated visual, sensor, and SAF devices. The approach presented will utilize event-driven changes to dynamically modify and manipulate multiple data sources in a central repository with the use of an event notification mechanism to ensure all client devices are updated to reflect the latest changes to the environment at run-time.

ABOUT THE AUTHORS

Kenny J. Hebert is the Product Manager for Content Creation at Presagis USA. In this role he is responsible for the technical direction and oversight of the Creator, Terra Vista, and SEGen Server product lines. Prior to assuming his current position as Product Manager, Mr. Hebert served as a Regional Manager and Application Engineer with TERREX Inc. Before joining Presagis, Mr. Hebert also worked for Northrop Grumman in Grafenwoehr, Germany producing synthetic databases for various DoD programs and as an Imagery Analyst for the National Geospatial-Intelligence Agency (NGA). In addition, Mr. Hebert holds a Bachelors and Masters of Science degree from the University of Southern Mississippi and has over 15 years' experience as an Intelligence Analyst with the US Marine Corps.

Dale Sexton is the Director of Product Management for Presagis. He has a Bachelor of Science Degree in Electronics Engineering. Mr. Sexton as over 27 years' experience working in all aspects of Modeling and Simulation products. Moreover, he has experience in several M&S positions including: Hardware Engineering, Systems Engineering, Software Engineering, Radar Engineering, Project Engineering, and Program Management with leading companies in North America.

Dynamic Synthetic Environments Through Run-Time Modification of Source Data

Kenny J. Hebert, Dale Sexton
Presagis
Richardson, TX
Kenny.Hebert@Presagis.com, Dale.Sexton@Presagis.com

INTRODUCTION

Realistic synthetic environments are a key component for the effectiveness of any simulation-based training. In order to achieve the necessary levels of realism required for effective training operations, it is vital that training participants be able to adapt their actions to a changing environment regardless of whether the change is the result of naturally occurring environmental effects, such as floods or erosion, or simulation driven events, such as bomb craters. This level of realism requires the ability to dynamically interact and manipulate the synthetic environment within a simulation during run-time. Achieving such realistic dynamic terrain requires both the accurate representation of terrain together with the ability to modify that terrain at run-time during a simulation.

While there has been extensive research aimed at resolving the issues associated with realistic dynamic synthetic environments, the main concepts of *dynamic terrain*, *destructible terrain*, and *deformable terrain* are used inconsistently throughout the literature. Therefore, in the interests of clarity, we will use the concept of *dynamic terrain* in this paper to refer to all aspects of a changing environment in a simulation, including the actual terrain itself as well as objects located on the terrain, whether they are buildings, other manmade structures, or objects such as vehicles. The concepts of *destructible terrain* and *deformable terrain* will be used in a more narrow manner that refers to specific subsets of dynamic terrain and only refer to changes being made to the underlying terrain and not to the interaction of objects located on the terrain. The continuous client-based approach to the dynamic modification of synthetic terrain that we are proposing in this paper focuses on destructible terrain and presents a method for overcoming inconsistencies in correlation across participating devices that can lead to ineffective simulation-based training.

Research efforts that directly address the ability to accurately simulate destructible terrain have, traditionally, used either appearance-based or physics-based approaches. If we considered the case of having to visualize a crater created by a missile in a virtual battlefield, an appearance-based approach could generate the destructible terrain either by substituting a real texture of a single crater or by generating smooth deformations. Conversely, physics-based approaches use physics based models that modify the elevation of terrain vertices at run time using natural physical models. While such physics-based methods do achieve higher fidelity visual deformations than appearance-based approaches, neither method takes into account the dynamic modification of original source data or other derived datasets used by client devices in a training event.

In the current M&S environment, most simulation-based training involves multiple applications and client devices that require more than just using natural physical models or attributing terrain polygons and visual images. For instance, while some applications use native Geographic Information System (GIS) source data, other devices use derived datasets. Therefore, in the case of destructible terrain generation, if one only uses an appearance-based or physical-based approach that simply modifies the surface geometry and visual representation to simulate a dynamic event—such as an explosion on a road—then there may be systems participating in the simulation that do not reflect these changes because they require an updated vector representation or an updated raster material layer. This leads to inconsistencies in correlation across the participating devices and, ultimately, to ineffective training.

The focus of this paper is to present a comprehensive approach for developing destructible terrain that ensures that multiple data types and formats can be dynamically modified and updated at run-time and that is generic enough to include multiple fidelities, including game-based effects and physics-based models. The proposed architecture for the dynamic modification of synthetic terrain data is designed to support correlated visual, sensor, and SAF applications. In particular, this approach uses event-driven changes

in a simulation to dynamically modify and manipulate multiple data sources in a central repository with the use of notification mechanisms to ensure that all client devices are updated to reflect the latest changes to the environment at run-time.

BACKGROUND AND RELATED WORK

Strategies for Creating Terrain Databases

The hand modeling of terrain is one strategy for creating databases for simulations and is best suited for close combat training operations and urban simulations. While hand modeling using a 3D modeling tool like Creator generates very high quality and very realistic terrains, the process can be very costly and time consuming and requires highly skilled artists.

Another strategy for creating terrain databases is to automate parts of the process using database generation tools like Terra Vista that employ parametric algorithms to produce large areas of dense detail. Such databases are suitable for air crew flight training, large scale ground simulations, and large area operations. Following this strategy, database generation tools ingest source data and apply processing rules and functions to that provide mapping from vector, elevation, imagery and 3D models into the various simulation databases. After processing, the terrain data can be output in a variety of correlated formats for both visualization and simulation. However, these large area terrain databases are static and may require a large amount of disk space. Additionally, because database generation tools are offline processes, any changes to the source data requires rerunning the database generation tool to regenerate the terrain and, thus, can be a very time-consuming.

Government Sponsored Initiatives to Support Dynamic Terrain

In order to populate dismounted infantry training simulations with real-world dynamic terrain and objects, the U.S. Army Research Laboratory (ARL) conducted early research into effective methods for computing and distributing dynamic terrain information in real-time distributed simulations (Thomas, 2003). Their research led to the development of the ARL Dynamic Terrain Server (DTServer), which delivered fair-fight, realism, and effective training by providing users with a unified terrain database across all simulators. The DTServer created this unified terrain database by first computing the effects of explosions and collisions on structures and then transmitting the results to client servers and updating those servers on the status of rubble entities in the simulation.

However, the DTServer does not run 'direct from source', but rather, runs using a terrain database, object database, and munitions database. The terrain database was used for ground clamping rubble, and multiple terrain databases can be loaded into the DTServer. For the object database, the DTServer can either function on an entire database or on specific objects within a database and processes objects in a database following a user-created datafile of object names. And, finally, the munitions database used by the DTServer contained such information as name, DIS enumeration, equivalent weight of TNT, and muzzle velocity that were all derived from open-source literature. To compute the effects of munitions on structures, the DTServer used table look-up and empirical formulas. Using a munitions effects table containing data for relevant munitions, the DTServer computed the effects and distributed them to compliant simulations across the network using custom distributed interactive simulation (DIS) protocol data units (PDU).

Another initiative was the US Army's GIS-Enabled Modeling and Simulation (GEMS) project which allowed for models and behaviors in constructive simulation applications to interoperate with an ESRI geodatabase. This approach uses a series APIs and software wrappers that map common functionalities and actions performed by M&S applications to the GIS application and data (Stanzione & Johnson, 2007; Wiesner, Brockway, & Stanzione, 2011). This approach allows for M&S applications to take advantage of the geo-spatial modeling capabilities of the GIS system, such as offloading terrain reasoning functions. However, in this approach, the M&S client applications do not use the GIS data as their internal run-time terrain representation.

Other research efforts on Dynamic Terrain in the Modeling & Simulation domain have also focused on improving the level of fidelity of those elements in a simulation located on top of the destructible terrain. For example, in their article entitled "Using Real-time Physics to Enhance Game Based Effects," Mann, Lyons, & De La Cruz (2008) implemented physics effects in training scenarios designed to help troops to develop judgment and decision making skills for dealing with an Improvised Explosive Device (IED) attack. While using physics models in training simulations to simulate such complex effects as those produced by munitions is not a new idea, what is new according to the authors is the capability to use such validated physics models in real time environments. In their paper, Mann et al. discuss a research effort sponsored by the Army RDECOM STTC to

"demonstrate the feasibility of using physics-based weapons effects models in a game engine and to develop an approach for optimizing the models for real-time response" (Mann et al., 2008).

Other research was conducted on leveraging the OneSAF Simulation Object Runtime Database (SORD) Gateway interface to implement dynamic terrain capability into OneSAF (Graffuis, Munyer, & Mculley, n.d.) One project required injecting physics effects into OneSAF and visualizing the results in a 3D renderer so that most modeling would be handled by OneSAF while the higher fidelity detonation effects would be determined and visualized elsewhere (Gaffuis, et al., n.d.). In order to model flyout debris from various urban area munition detonations, another project used calculated rubble piles and their volume as input to OneSAF to effect aspects of the scenario and visualize the results. For both of these projects, the researchers chose ARA's Real-time Physics Effects Library (RPEL): first to evaluate the outcome of the munition detonations and second to determine the amount and size of the rubble.

Similarly, Browning, Adkinson, Borkman, & De La Cruz (2009) introduced a methodology that incorporated an alternate environmental representation into a physics engine that would allow users to achieve a terrain representation level appropriate for military training simulations while still having access to the more complex physical interactions available with a robust physics engine. In particular, they used the Live Terrain Format (LTF) to power the terrain representations associated with the Nvidia PhysX physics middleware solution.

Their research worked to address two main issues: the correlation of terrain between simulators and the ability to dynamically modify terrain. Their research focused on using queries to obtain useful information from terrain databases with these queries typically taking the form of collision detection or line of sight testing. In particular, they explored the benefits and limitations of integrating LTF and PhysX in regards to queries against the terrain. Their research focused on four attributes of LTF and PhysX: correlation with existing formats, maximum terrain size that can be loaded and run in real-time, performance of line of sight queries, and performance of collision detection against the terrain.

Commercial M&S and Gaming Initiatives to Support Dynamic Terrain

In 2003, Kamat & Martinez reported on their development of ViTerra, a tool that automatically generated photorealistic, digital, 3D terrain databases to represent construction jobsites in visualizations of discrete-event construction simulations using detailed topographical imagery—such as Digital Elevation Map (DEM)—and aerial imagery—such as National Aerial Photography Program (NAPP)—data that are available in digital form from government and private sources. By depicting deformations to the loaded terrain in response to construction operations (such as earthmoving and trenching), ViTerra also implemented animation methods to describe the evolution of virtual jobsites. According to the authors, there are numerous challenges related to designing simulation-driven methods that automatically generate and then visualize the deformation and evolution of jobsite terrains in response to common construction processes in smooth, continuous, dynamic, 3D virtual worlds. First, while digital elevation and aerial imagery data is readily available for the entire United States and several other parts of the world, this data is archived in several different digital formats at varying levels of detail.

Another challenge is the fact that the resultant locally deformed shape and appearance of a virtual terrain in response to an animated elemental construction task (like digging) cannot be determined a priori by animation-authoring processes (e.g. discrete-event simulation models) or by the visualization engine itself. In other words, the deformation that a virtual terrain undergoes in response to an animated construction task depends on the attributes of the virtual equipment and on the amplitude of the motion of its components while undertaking that task. Therefore, as the authors explain, "the computation that determines the shape of an evolving terrain in response to animated construction processes must thus be performed during animation in real-time and is solely the responsibility of the visualization engine" (Kamat & Martinez, 2003).

The final challenge concerns how computationally intensive it is to maintain and render virtual terrains because terrains are typically composed of several thousand textured polygons. Rendering dynamic terrains in real-time in 3D animation at interactive frame rates usually requires limiting the number of polygons that the graphics pipeline needs to display without compromising the detail of the terrain, especially in simulations where the terrain is seen and manipulated at close range. In such instances, the simplification of the terrain database to improve performance cannot compromise the fidelity of the terrain's appearance.

To address these challenges, Kamat and Martinez first implemented techniques to automatically generate terrain Computer Aided Design (CAD) databases by

combining data from disparate elevation and imagery data sources into a unified, consistent data set and then investigated feasible data structures to internally maintain and locally manipulate (on demand) the generated terrain databases. In addition, they designed methods both for computing the deformations that virtual terrains undergo in response to animated construction processes and for applying those deformations to the terrain databases in real-time.

In 2001, Davison and Wang also proposed a way to add significant deformable terrain game play features to an existing Real-time Strategy (RTS) Game. From the terrain generation perspective, developing an RTS game with incorporated terrain deformation features raises a number of challenges in both technique implementations and system design, including how to define the terrain, store the terrain internally, render the terrain, and deform the terrain. They presented a system with the ability to deform terrain for an RTS Game with general PC hardware specifications, including being able to simulate such effects as raising and lowering the ground, creating a large chasm, or leveling the terrain.

CHALLENGES TO CURRENT SOLUTIONS

To date, solutions for deformable/destructible terrain have dealt primarily with the modification/animation of objects residing on the terrain and not modifications of the terrain itself. When the U.S. Army Research Laboratory (ARL) conducted early research into effective methods for computing and distributing dynamic terrain information in real-time distributed simulations, their research was concentrated on standardizing the communication of the events that caused the destruction and ensuring correlation between clients for the resultant effects on structures in the environment. In their paper, Mann et al. (2008) discuss efforts that are restricted to those dynamic elements of a synthetic environment that are located on top of the terrain. While feature destruction has value to add to training, feature destruction without underlying terrain deformation can offer degraded training.

Historically, two methods have been employed to implement dynamic synthetic environments; modification of representations of static terrain stored in memory at runtime and on time-varying geometry, also called switch state, manipulation. Static terrain modification requires that the entire terrain is loaded into memory where it can be modified during the exercise. In the case of ViTerra, as reported by Kamat & Martinez, the terrain had to be first loaded into the renderer's memory prior to any modification. Therefore, any modifications to the database must be based on the current representation of the environment loaded into memory and are unable to incorporate new data after the start of the exercise. Additionally, modification to in-memory terrain is dependent on the available memory on each and every client that is part of the exercise.

Usage of Level of Detail (LOD)

In many dynamic terrain configurations that use static terrain databases, any modification of the database typically occurs on a single LOD of the database and defeats the purpose of having multiple LODs in the exercise. As reported by Kamat and Martinez, the simplification of the terrain database to improve performance cannot compromise the fidelity of the terrain's appearance. This is the basic construct behind the usage of LODs in a database; to improve the fidelity of the database dependent on the perspective of the player. Therefore, the inability to allow each of the clients to choose which LOD to modify becomes a training degradation issue for that client.

Data Persistence

Any modifications made to the in-memory representation of a synthetic terrain database are performed as a 'snapshot' of the current change in memory and any modifications to the terrain are not written back to the original database. This inability to write back the changes limits the ability for persistence between different runs of the same or similar mission. Without this capability, degraded training between exercises can result.

In cases where each client is operating on the same database (loaded in memory locally), the inability to write back these changes to the original database, along with the inherent difficulty in synchronizing a distributed exercise, results in numerous variances between the same exercises. When a static database can be modified in memory and the lack of correlation and persistence is acceptable, temporal changes, such as the digging of a trench, appear instantly as opposed to varying over a specific time to reflect real world events. Modifications performed on databases loaded in memory tend to concentrate on elevation and imagery and ignore vectors and material changes which prove harder to update at runtime.

Correlation in a Distributed Exercise

For distributed exercises, there is no means to ensure that each client modification is performed in the same fashion, without which, correlation between clients cannot exist. According to Browning, et al., (2009), due to a lack of overlap between supported terrain

forms, a large variance in results communicated between simulations can result. During their research efforts utilizing PhysX and LTF, they discovered that these formats were heavily dependent on their own representation in order to bring in terrain even though LTF was capable of being correlated with a number of other formats. Add in the complexity of dynamically modifying each client's in-memory representation of the terrain and the correlation issue increases in complexity. While efforts in the past have concentrated on standardization of the communication protocols for dynamic events, few have taken into consideration the need for standardization on the modification of the underlying terrain.

REQUIREMENTS OF AN OPEN ARCHITECTURE FOR RUN-TIME DESTRUCTIBLE TERRAIN

In order to provide the maximum level of flexibility and accessibility for simulation devices to implement a run-time dynamic terrain capability from source, it is important to first identify the necessary requirements for said architecture. It is understood that each training device will usually have unique requirements specific for its given use-case. However, the requirements being proposed for an open architecture for run-time destructible terrain provide the maximum flexibility for cross domain functionality, fidelity, and performance.

Open Source Formats

Current simulation and visualization applications in the Live, Virtual, and Constructive (LVC) training domains rely heavily on the synthetic terrain representation. Usually, these synthetic environments are compiled into a static terrain database tailored for a given simulation application or use-case. Often times, these terrain databases are in a proprietary format or source that limits the cross-domain interoperability, exchange, and re-use across the LVC domain. Therefore, the first requirement proposed is the direct use of open GIS formats and standards for run-time use and dynamic modifications.

Unlike other approaches that utilize pre-compiled static databases and/or provide run-time *access* to GIS source data using APIs and software wrappers, we propose that the simulation clients use the GIS data as their native terrain representation at run-time. Assuming disparate simulation clients, it is also important to have a common understanding of the data types, data models, data structure, data attributes, and other factors to ensure there is correlation across the client devices and reduce any unnecessary conversions or data mappings. Standardized GIS source data ensures that all of the client devices can easily access and understand the semantics of the source data at run-time without any additional conversions or translations. For instance, GIS elevation data can be represented in various formats such as DTED, DEM, GeoTIFF, BIL, etc. Vector data also comes in a variety of formats (such as VMAP, Open Street Map, DFAD, ShapeFiles, etc.) where each vector type has a unique attribute schema and file structure. If there was not a common standard to the GIS source repository, each of the client devices would have to support and interpret an endless number of formats, projections, data schemas, etc., which would limit the ability for efficient run-time simulation.

Therefore, to support an open architecture where simulation clients are running a simulation 'direct from source', it is important for all of the disparate simulation clients to access and run from a commonly agreed upon standard for the GIS source data. This would include defining the source data types supported as well as the attribution schemas and geographic projections. Additionally, the source data should be in industry standard open formats that are easily accessible by a wide range of clients. These include formats such as ESRI ShapeFiles for the representation of the cultural and geographic features, GeoTiff or JPEG 2000 for Imagery and Elevation, OpenFlight for 3D models, and RGB for textures.

Any deformable terrain solution shared across multiple disparate clients should also leverage current interoperability standards for any type of notification of deformation. Given that the current standards, DIS and HLA already contain PDUs that provide the necessary data to model a detonation event, it would make sense to leverage these where possible for detonation related deformations. Other types of terrain deformation not related to a detonation event would need to be considered as an extension to existing HLA as well as supported DIS PDUs.

Composable Data Structure (Independent File Access and Modification)

In order to support the widest range of client devices in the most efficient manner, the run-time source data should also be highly composable allowing the individual client devices to select only those data types that are required for their given use-case. A highly composable structure allows for independent components of the source data to be selected and assembled in a variety of combinations to satisfy the specific requirements of the simulation client. This composability of the source data also provides maximum efficiency and optimization by allowing the client device to only load the source data that is

required and relevant for its given use case, thus saving valuable system resources and memory for other usage.

Scalable Data Access and Modifications
In addition to allowing the simulation client devices to independently select and modify the individual data sources that are relevant for its specific use-case, it is also important for those datasets to be arranged in a scalable structure that allows the client device to load only those parts of the dataset that are necessary to meet the spatial and fidelity requirements of the operation.

While a composable structure allows clients to select the types of data needed for a given simulation, a scalable structure allows client devices to select the levels of fidelity needed within those data types. With a scalable data structure, the independent client devices are able to better manage the memory and storage requirements by only loading the range of resolutions needed for a specific area or event. For instance, if a large crater is dynamically generated in the terrain skin at run-time, a ground vehicle will require an update with the highest available LOD for the elevation and imagery datasets at the location of the crater. Conversely, a fixed-wing asset flying over the terrain may only require an update to the texture and material representation for that same crater at much lower resolutions. With a properly structured dataset, the scalability of the spatial and fidelity extents should be near infinite with world-wide coverage and resolutions that are only constrained only by the capabilities of the client device.

Persistent Data Access and Storage
In order to ensure correlation across multiple disparate simulation devices, as well as persistence of the data between simulation runs, it is necessary for any of the dynamic events or changes to the source data be written back to disk. Although many run-time dynamic architectures access and modify the in-memory terrain data for efficient processing without latency, there are few that actually save the source data modifications back to disk. Therefore, once the simulation run ends, or the source data is purged from memory, any changes made to the original terrain data are lost. In addition, notification mechanisms should be in place to ensure that when a change is made to the source data, all of the client devices are made aware of those changes and are able to reload the modified data as required.

Open and Extendible Architecture
Another primary requirement stressed by the authors is the implementation of an open architecture and API for dynamic terrain capabilities that is scalable, configurable, and adaptable. The motivation for this requirement is to allow for the easy integration of additional capabilities and modules by enhancing or customizing the system to satisfy domain or simulation specific needs. This allows the flexibility to leverage only those components that are relevant for a given simulation or replace entire components without impacting the overall architecture. For instance, if realistic environmental destruction is required for a ground-based simulation, the integration of a physics engine can be added to the overall architecture to meet the simulation requirements. In addition, an open architecture promotes wider adoption and reuse throughout the M&S Community by providing a readily available and accessible framework that can be quickly adapted to meet various training needs, thus lowering the development/integration time and costs.

PROPOSED ARCHITECTURE FOR IMPLEMENTING DYNAMIC TERRAIN THROUGH RUN-TIME MODIFICATION OF SOURCE DATA

The proposed architecture for the dynamic modification of synthetic terrain data is designed to support correlated visual, sensor, and SAF applications. In particular, the approach presented uses event-driven changes in a simulation to dynamically modify and manipulate multiple data sources in a central repository with the use of notification mechanisms to ensure that all client devices are updated to reflect the latest changes to the environment at run-time.

GIS Source Data Repository
To meet the requirement of direct usage of open GIS formats, we propose the use of a standardized GIS Source Data Repository that contains all of the synthetic environmental data stored in open formats and consolidated into a single data repository. The structure of the GIS datasets within the repository are organized in such a way that allow for efficient run-time publishing and independent file access and versioning for dynamic and persistent modifications to the source data. Specifically, the GIS data is organized into *tiles, layers,* and *Levels of Detail (LODs)* that provide worldwide coverage at extremely high resolutions and fidelities.

With this structure, the GIS data *tiles* refer to the spatial extents for which the data is organized. Each geodetic tile (bound by latitudes and longitudes) contains independent *layers* of GIS data that represent the synthetic environment (e.g., imagery, elevation, 3D models, etc.). Additionally, each layer is sub-divided into LOD representations, ranging from extremely high

resolutions to very coarse representations, which are critical for run-time performance.

Data Server

To allow client devices to access the GIS source data located in the source data repository, without requiring extensive modifications to each client, the authors propose the usage of a 'Data Server(s)'. The Data Server will provide runtime access to the source data stored in the repository by retrieving the appropriate data type at a given location for a specific LOD or range of LODs. The Data Server should also cache the requested data and adjoining areas in local memory to increase performance by limiting read/write disk access thereby ensuring optimal runtime performance.

In addition, the Data Server should support multiple simulation clients, particularly when different clients are operating within the same geographic area. Thus eliminating the need for each client to independently access the source repository or maintain separate data servers that are accessing the same data from the repository.

Dynamic Update Engine

In order to perform the actual source data modifications and transform dynamic events received at run-time into source data modification commands, the use of a 'Dynamic Update Engine' is proposed. The purpose of the Dynamic Update Engine would be to receive and process commands to modify the specific data types, layers, and LODs for a given location during run-time. The Dynamic Update Engine would also notify clients of any relevant source data changes to facilitate update of terrain representations specific to the affected tiles, layers, and LODs. In addition, the Dynamic Update engine would need to write the modified source data files back to the repository as new versions of the datasets.

To address persistence between simulation runs, changes made by the Dynamic Update Engine would need to be applied in the order in which they occurred (to ensure causality with respect to events). The ability to write some or all of these changes should be based on user-defined preferences to ensure persistence is at each client's discretion (e.g., temporal parameters may be set when rapid dynamic changes occur in the simulation but only the end result needs to be saved to the repository).

As supported by Lagace (2010), the proposed architecture meets all of the requirements previously mentioned and provides an open and scalable architecture that is based on open source GIS data in a structure that is composable, scalable, and designed for efficient and persistent run-time simulations. The components of this architecture are depicted in Figure 1 as described above.

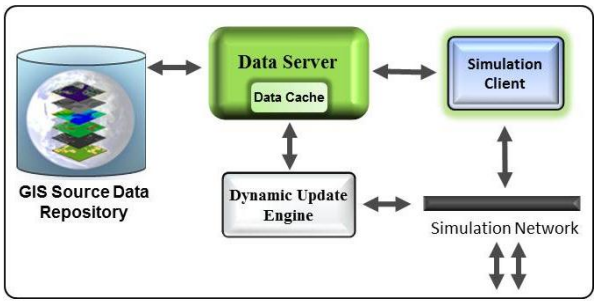

Figure 1. Components of the Run-Time Architecture

Dynamic Event Process Workflow

To illustrate the process workflow using the proposed architecture, we will use the example of a driver training device and helicopter flight training device participating in a simulation federated with constructive ground vehicles and other platforms.

While there are many variations that can be made to the proposed architecture, for this example we chose to implement the structure depicted in Figure 2. As depicted, all of the client devices access the GIS data layers and 3D models in the GIS Source Repository through the Data Server. When the client registers to the Data Server, the relevant data layer at a given location and the relevant LODs within a given range are extracted from the Source Repository and cached into memory by the client device.

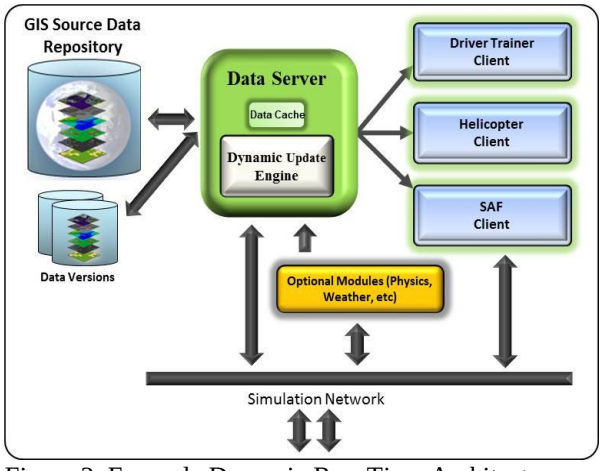

Figure 2. Example Dynamic Run-Time Architecture

Unlike the previous diagram, where the Dynamic Update Engine was an independent module in the process, we chose to include the Dynamic Update

Engine with the Data Server. This allows the Dynamic Update engine to make local modifications to the source data already in memory by the Data Server and multicast the specific information and updates needed by the clients. Once the clients receive notification that a change to the source data has occurred, they simply update and refresh the affected LODs or data tiles that are already in the client's memory cache. When all of the changes are made to the in-memory data sources, the Dynamic Update Engine performs background write capabilities of the modified source data to the Source Data Repository. The Dynamic Update Engine also provides access to additional destruction modules (such as a physics engine or weather module) via the network.

In this example, the virtual driver trainer, which requires extremely high ground fidelity and resolutions for the visual, is using elevation source data at a resolution equal to 1m grid post spacing, imagery at .25m, and 3D model LODs with complex geometric and texture representations. Conversely, the helicopter client is accessing data from the same Source Data Repository as the driver trainer but requires additional data layers and multiple LOD ranges depending on the altitude of the platform. In this case, the helicopter is providing air support for the driver trainer and is operating at an altitude that needs much lower resolutions/LODs. Therefore, the helicopter client is using elevation data equal to 27m, imagery at 14m, raster material datasets at 14m, and 3D models with lower geometric and texture representations. In addition to the imagery, elevation, raster materials, and 3D models used by the driver and helicopter trainers, the constructive clients also require the vector data layers for calculations such as entity route planning.

Figure 3. Dynamic Crater in the Synthetic Environment

In this scenario, an IED detonates in the middle of a road as the driver trainer approaches and creates a large crater that is approximately 15m wide and 3m deep impacting the simulated environment (depicted in Figure 3). The Dynamic Update Engine receives the detonation event from the constructive client and translates that event into source data modification operations. Since the source data is already in memory from the Data Server, the Dynamic Update Engine applies the following changes to the source data in memory starting with the highest LOD and propagating to all of the lower LODs affected by the event:

- The 1m elevation data tile is extracted and a deformation is made at the location of the explosion that is 15m wide and 1m deep. The subsequent lower LOD tiles are also modified up to the LOD representing 15m (since the crater is 15m wide, there is no need to modify the elevation beyond the 15m LOD since those changes will not significantly impact the terrain profile).
- The .25m imagery data tile is extracted and a texture mask is applied at the detonation site that changes the visual appearance of the road from asphalt to fresh soil and rubble. The lower LOD representations are created by resampling the .25m representation into coarser resolution image files up to the LOD that is impacted by the change.
- The .25m raster material data tile is extracted and a texture mask is applied that correlates to the visual image tile and modifies the property of the data pixels from asphalt to fresh soil. Lower LOD representations are created up to the LOD impacted by the change.
- The vector data containing the road network layer is extracted from the cache and the linear vector impacted by the IED is modified by inserting vertices that correspond to the outer edges of the crater into the linear vector and removing that section of the vector. Thus, creating a break in the road network at the point of impact.

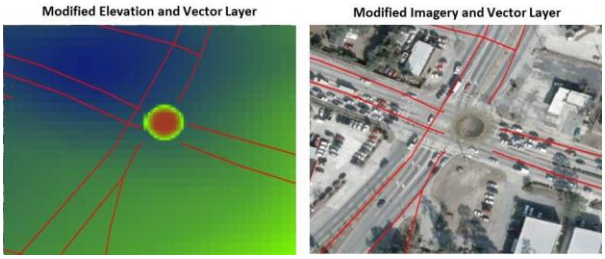

Figure 4. Example of Imagery and Road Network Layers Before and After the Dynamic Event

When the changes are made to the appropriate source data type, a notification is sent by the Dynamic Update Engine to all of the client devices that are within the pagable geographic extents of the changes. The client devices receive a notification that the source data has changed and only updates the in-memory data types and

LODs that are relevant. In this example, the driver trainer updates the imagery layer at .25m and the elevation layer at 1m resolutions (Figure 4). The helicopter client only needs to update the imagery and raster material layers at 14m since the changes to the elevation are not relevant for his current altitude (that is using 27m elevation data). Similarly, the constructive client does not require the imagery dataset but does utilize the vector data. Therefore, the constructive client loads the elevation and raster material layer at 1m and the vector road network layer (Figure 4).

Once the changes are made to the cached source data by the Dynamic Update and Persistence Engine, all of the modified data files are written back to the Source Data Repository as separate versions of the original data files. When multiple events occur in the simulation at the same location, the changes are made iteratively to the most recent version of the impacted data type so that changes are cumulative and persistent.

Because the Dynamic Update Engine only modifies the relevant source data types at the relevant LODs, and the source data itself is optimized for run-time, the total amount of data required for read and write operations to/from the Data Server and Source Data Repository is quite small compared accessing the full native source file. In this example, the imagery data was modified in the LODs from .25m resolution to 15m resolution (when the crater dimensions exceed the pixel resolution) for a total of 7 LODs. Even though the helicopter simulation client is using the raster material data layer at 14m resolution when the detonation occurred, the raster material LODs from .25m to 15m were still modified to ensure correlation with the imagery as the helicopter pages in higher resolution data. The elevation data tiles were modified in the LODs from 1m to 15m resolution for a total of 4 LODs and the road network vector only had to be modified at a single LOD.

Furthermore, each of the raster data tiles (imagery, elevation, and raster material) at each LOD consists of 1024x1024 data tiles; yet each has a specific byte structure to support the given data function. Therefore, while the spatial size of the data tiles remains the same, the file size of each tile differs depending on the data type. In this example, the total amount of data accessed, modified, and written back to the repository only consisted of 18MB of total data as depicted in Table 1.

Table 1. Example of File Size and # of Modified Data

Data Type	File Size of Each Data Tile	Number of LOD Tiles Affected	TotalSize of All Affected Tiles
Elevation	4000 KB	4	16,000 KB
Imagery	300 KB	7	2,100 KB
Raster Material	21 KB	7	147 KB
Road Network	50 KB	1	50 KB
		Total Space	**18.3 MB**

As demonstrated, the proposed architecture allows for the dynamic modification of synthetic terrain data is designed to support correlated visual, sensor, and SAF applications. It also provides the capabilities necessary to meet all of the previously outlined simulation requirements while remaining flexible enough to support future requirements.

BENEFITS OF AN OPEN ARCHITECTURE FOR RUN-TIME DYNAMIC TERRAIN

Many of the current simulation systems used across the LVC domain utilize pre-compiled static terrain databases that try to encompass all of the relevant terrain features into a single file. As previously highlighted, this approach has many different limitations (particularly for dynamic environments) that often result in trade-offs between the fidelity, resolution, and size of the synthetic environment to meet the limitations of the client's system resources. Therefore, the architecture and process proposed offers many advantages over the pre-compiled database paradigm by accessing and modifying source data at run-time.

One of the most prevalent benefits of having structured source data, with independent layers and LODs, for run-time dynamic modification is the portability and composability of the data and processes. This provides the ability for the architecture to be tailored to meet different simulation requirements and devices. For instance, since different data types (imagery, elevation, etc.) are stored in independent layers with independent LODs, a client device with limited hardware and memory (such as a mobile platform) will only use the required data types at the required resolution(s) for any given mission. If the client does not require vector data for route planning or raster material data for traffic analysis, the client does not have to load those datasets and thus, can focus the system resources and memory on performing tasks that add value to the target usage. Compared to many of the traditional approaches that compile all of the source data content into a static terrain database (which can be quite large and require specialized hardware resources), segregating the data into their respective data layers and optimizing the

structure for run-time performance, provides greater flexibility and possibilities for LVC simulations.

Furthermore, in today's training environment, there are many cases where dynamic changes to the environment must occur in near-real-time and be distributed to all of the participating client's without affecting correlation. Structuring the source data into independent tiles, layers, and LODs reduces the amount of time needed to access the data, modify the data, utilize the data on the client device, and write the changes back to disk (e.g., latency). For instance, if a small dynamic event occurs in a simulation (such as the IED explosion in our example) and the original imagery source covered an entire geocell at 1m resolution, without an efficient tiling structure, the client device would be required to access the entire image file just to make changes to very small area. Not only does this create a large burden on the system memory requirements, as previously mentioned, it also takes longer for the operation to be completed and the client devices updated. Therefore, to reduce the amount of latency when a dynamic event occurs in a run-time environment, it is important for the client devices to only access the type and size (spatially) that is needed to perform the operation.

Another important benefit for the proposed architecture and process is the ability to maintain persistent copies of the modified source data files as temporal versions in the source data repository. Many of the current approaches that implement dynamic changes at run-time today do so by making modifications to the in-memory representations of the synthetic environment. In these cases, any changes made to the in-memory synthetic environment only remain prevalent during the simulation run and are not persistent between simulation runs without regeneration of the event or after the simulation ends. While the approach presented in this paper also accesses the source data in-memory by the Data Server (for faster updates to the clients), the Dynamic Update Engine provides the ability to write the modified source data back to the Source Data Repository and provide persistent data access between simulation runs and after the simulation ends. Furthermore, the Dynamic Update Engine can be configured to perform the file write functions to the Source Data Repository in the system background so as not to impact run-time performance of the Data Server or client devices.

Since the proposed architecture is a modular and open architecture that uses open source GIS data and communication protocols, it provides the flexibility for other 3rd party applications and processes to easily be integrated. Additional capabilities, such as physics based destruction engines and weather servers, can be integrated into the workflow with their dynamic changes propagated back to source data.

Lastly, since the proposed architecture is accessing and modifying the GIS source data at run-time, *all* of the data types that are affected by a dynamic event modified and updated in the Source Data Repository. When dynamic events occur in the synthetic environment, those events often result in changes to more than just the elevation and imagery. A simple change to the imagery data, for example, may have client devices that also require changes to the associated raster materials, light-maps (for night-time operations), and sensor maps to maintain correlation. Accessing and modifying the source data provides the flexibility for a wide number of applications and client devices, which utilize these data types, to access the dynamic changes without drastic changes to their client system.

CONCLUSIONS

In order to meet the constantly advancing requirements and needs of the Modeling and Simulation Community, the authors feel it is important to provide an architecture that is easily accessible and customizable that utilizes open source formats and protocols. As such, a methodology was presented that provides dynamic and persistent modifications to GIS source data at run-time in a scalable and composable manner to accommodate the widest range of simulation clients across the LVC domain.

REFERENCES

Browning, D., Adkison, J., Borkman, S., & De La Cruz, J. (2009). PhysX Extended: An Integrated Real-Time Physics and Terrain Solution. Proceedings from the *Interservice/Industry Training, Simulation & Education (IITSEC) Conference.* Orlando, Fl.

Davidson, C. & Tang, W. (2001). Deformable Terrain Generation for Real-time Strategy Game. Proceedings from the *EUROGRAPHICS 2001 Conference*.

Graffuis, S., Munyer, R., & McCulley, G. (n.d). Enhancing Simulations with Improved SORD Gateway Capabilities. Retrieved March 12, 2012 from http://www.stackframe.com/documents/SGW/10S-SIW-047.pdf

Kamat, V., & Martinez, J. (2003). Automated Generation of Large-Scale Dynamic Terrain in 3D Animation of Simulated Construction Processes. Proceedings from the *CONVR2003 Conference on*

Construction Applications of Virtual Reality. Virginia Tech, Blacksburg, VA.

Lagace, M. (2010). Dynamic Synthetic Environments and the CDB. Proceedings from the *IMAGE Conference*. Scottsdale, AZ.

Mann, J., Lyons, J., & De La Cruz, J. (2008). Using Real-time Physics to Enhance Game Based Effects. Proceedings from the *Interservice/Industry Training, Simulation & Education (IITSEC) Conference.* Orlando, Fl.

Stanzione, T., & Johnson, K. (2007). GIS Enabled Modeling and Simulation (GEMS). Proceedings from the *27th Annual Esri International User Conference*. San Diego, Ca.

Thomas, A. (2003). *The U.S. Army Research Laboratory Dynamic Terrain Server* (ARL Publication No. ARL-TR-2962). Aberdeen Proving Ground, MD. Retrieved April 20, 2012 from http://www.arl.army.mil/arlreports/2003/ARL-TR-2962.pdf

Wiesner, B., Brockway, D., & Stanzione, T. (2011). Open Streaming Terrain for Modeling & Simulation. Proceedings from the *Interservice/Industry Training, Simulation & Education (IITSEC) Conference.* Orlando, Fl.