DEVELOPING A FRAMEWORK FOR IG-INDEPENDENT PC-BASED DYNAMIC TERRAIN

Tim Woodard
Diamond Visionics
Vestal, NY

Graham Upton
Diamond Visionics
Vestal, NY

Rita Simons
Simulation, Training and Instrumentation Command (STRICOM)
Orlando, FL

ABSTRACT

Dynamic terrain is the ability to modify a terrain database in real-time. In the past, effects such as explosions did not alter the surface topology of the terrain by creating craters. The tracks and tires from vehicles did not leave ruts in areas with soft soil. This limited the training value of ground-based visual simulation systems. Dynamic terrain has become an increasingly important requirement for realistic ground-based simulation. The simulation of terrain deformation can be applied in training situations that involve vehicles causing ruts in soft soil, munitions generating craters, and the generation of trenches and berms by engineering forces.

Dynamic terrain has often been implemented using "tricks". For example, rather than placing an actual crater in the terrain as a result of a mine detonation, most simulations simply place a picture of a crater without changing the elevation of the affected terrain. In systems that actually implement dynamic terrain, they are not intended for warfighter-in-the-loop real-time visual simulation, they often depend on a proprietary IG, or are dependent on a special terrain database structure. This precludes their use with existing simulators using other visual systems. It also precludes the use of dynamic terrain in situations where heterogeneous simulators need to interact over a network.

Under the STRICOM SBIR program, Diamond Visionics has been tasked with developing a PC-based dynamic terrain solution that can be integrated with components of existing simulation systems. This included various IG, vehicle, networking, and SAF systems - each potentially running on different platforms.

This paper focuses on the challenges we faced in developing a flexible dynamic terrain solution. Specific topics discussed include software frameworks, the real-time modification of a terrain database, the realistic IG-independent visualization of terrain modifications, and the distribution of modifications to other networked simulators.

ABOUT THE AUTHORS

Tim Woodard is a lead software developer at Diamond Visionics Co. specializing in advanced C++ and real-time PC-based visual simulation. He has developed software to support high-speed deformation of visual databases on low-cost PC-based platforms. This technology has been used in laparoscopic surgery simulation and a dynamic terrain framework that has been integrating with existing IG system. Tim has completed two years of undergraduate study in the Honors Program at Heidelberg College in Tiffin, Ohio studying Computer Science and Mathematics.

Mr. Graham Upton is the Director of Engineering at Diamond Visionics Co. in Vestal, NY. He has 27 years of experience in systems engineering and management in the simulation industry. He holds a degree in Aeronautical Engineering from Southall College of Engineering in London, England. As a design engineer for Link Flight Simulation for over 21 years, he has been involved in the flight and ground vehicle products as well as in Air Traffic Control simulations. He has led R & D teams to develop new products in this fast-paced industry, and has managed both Military and Commercial programs. He has authored many papers in areas of simulation and training.

Rita Simons is currently a Visual Systems Engineer at STRICOM. She is the COR on the Low Cost PC Based Real-Time Dynamic Terrain Phase II Plus SBIR. She earned a Bachelor of Science degree in Electrical Engineering from the University of Central Florida (UCF) in Orlando, Florida and is currently pursuing a Master of Science degree in Simulation, in the Industrial Engineering Program at UCF.

DEVELOPING A FRAMEWORK FOR IG-INDEPENDENT PC-BASED DYNAMIC TERRAIN

Tim Woodard
Diamond Visionics
Vestal, NY

Graham Upton
Diamond Visionics
Vestal, NY

Rita Simons
Simulation, Training and Instrumentation Command (STRICOM)
Orlando, FL

INTRODUCTION

In visual simulation there is a plethora of system variation. Each system may be composed of unique hardware and proprietary software. Also, there is a wide variation in the fidelity and computational capability of each system. Given such a wide deviation in the architectures and abilities of fielded hardware and software, an implementation of dynamic terrain must be flexible if it is to be practical.

Diamond Visionics has been tasked with developing a dynamic terrain solution under the STRICOM Small Business Innovative Research (SBIR) program to address this requirement. The result of our work is an implementation that can be integrated with the components of *existing simulation systems*. This includes various image generators (IGs), vehicle dynamics systems, networking protocols, and Semi-Automated Force (SAF) systems - each potentially running on different platforms.

This paper focuses on the challenges we faced in developing a flexible dynamic terrain solution. Specific topics discussed include software frameworks, the real-time modification of a terrain database, the realistic IG-independent visualization of terrain modifications, and the distribution of modifications to other networked simulators.

BACKGROUND

In 1997, technology was being developed to support the visualization of laparoscopic surgery for use in medical training simulation. A complete solution required the ability to deform organs and tissue in the visual database in real-time, complete with a host of special effects including wetness, tool-surface reflection, and complex lighting. What was unique about this technology was that it did not require a high-end computing platform, but rather utilized COTS PC-based hardware.

Seeing that this technology had potential in other areas of visual simulation (namely, ground based simulation), these developments led to an initial implementation of dynamic terrain that was demonstrated at I/ITSEC 1997. This initial demonstration consisted of a bulldozer that could dig and dump soil and a tank that caused soil compression due to its weight. The deformation was limited to a pre-defined area of the database.

This initial dynamic terrain demonstration generated interest within the military simulation community, showing promise that a PC-based dynamic terrain implementation was possible. In 1998, we responded to an SBIR Phase I solicitation to propose a solution for PC-based real-time dynamic terrain and was awarded a Phase I contract to explore possible implementations.

Figure 1 - Phase I Option Demo Results

By the end of the Phase I Option we had demonstrated that PC-based real-time visualization of dynamic terrain in a distributed environment was possible (see Figure 1), but we acknowledged that our approach had a number of limitations. Namely, it was dependent on our own IG, and the algorithms that it used for

modifying the visual database could not be easily generalized. This precluded the integration of this technology with existing IGs. A practical solution would need to allow a dynamic terrain implementation to be useable in a wide range of existing simulation systems while requiring minimum integration effort.

In Phase II, these limitations were addressed by significantly enhancing the solution developed under Phase I and Phase I Option. Rather than developing YAIG (yet another IG) or require that dynamic terrain be re-implemented for each existing IG, it was decided that a framework approach would be more practical.

The Phase II effort included improvements to the overall architecture as well as the underlying algorithms used to support the generation, visualization, and distribution of terrain modifications. For the architecture, a C++ framework was developed to provide the level of flexibility necessary to allow the integration of dynamic terrain with a wide variety of existing image generation systems. While the technology and concepts related to framework architecture are not new, the application of this approach in implementing dynamic terrain is.

FRAMEWORK-BASED ARCHITECTURE

One of the biggest challenges in developing a framework is that they are significantly different than traditional application programming interfaces (APIs). An API is usually a collection of autonomous functions that can be invoked by a host application (see Figure 2). An API does not strictly impose the order in which its functions should be called. Rather, it is up to the developer of the application to determine the flow of control and call the API functions in the proper sequence. A framework is a "semi-complete" application architecture that defines the flow of control between the customizable components of a system (see Figure 3). A framework is customized for a particular application by substituting the needed components at specified locations in the framework. A framework only knows about the external interfaces of the components, not the internal implementation details.

With a framework, there is an inversion of communication. A framework calls functions in the user-provided components instead of the traditional approach where the user program calls functions in an API. This is the so-called "Hollywood Principle", i.e., "don't call us, we'll call you."

Benefits, Disadvantages and Applicability

The purpose of a framework is to provide the flow of control that is common to all applications that may use it. The benefit is that the amount of work required to develop a new application based on a framework when compared to using a traditional API is significantly reduced because the flow of control does not need to be re-implemented for each application. Frameworks therefore can greatly increase the potential for reuse.

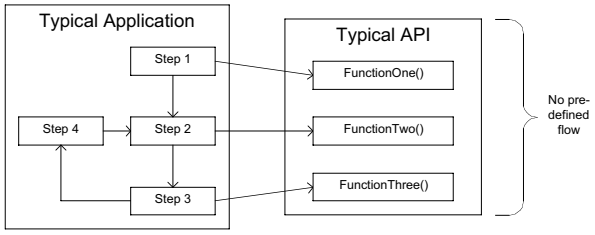

Figure 2 - Application Using an API

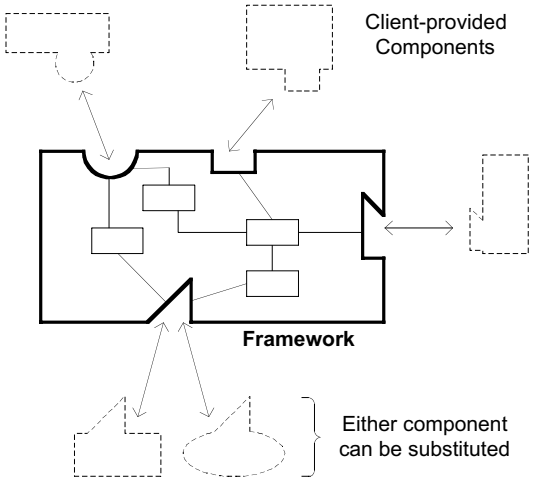

Figure 3 – Application Using a Framework

Frameworks are very flexible in that they allow user-defined customized behavior through the use of user-provided components. A framework may also optionally include default behaviors to make initial integration efforts easier. As the integration of a framework progresses, each default behavior can be overridden as necessary to fulfill the application's requirements.

Despite these advantages, frameworks have the disadvantage of being relatively difficult to develop properly [Beedle 1998]. First, expertise in the domain is required. Often, gaining this expertise takes years, as there may be many subtleties in a given domain. Also, an understanding of the user community and their (sometimes changing) priorities is necessary. Ongoing communication with the user community is critical to determine exact requirements and expectations. Advanced object-oriented (OO) modeling experience is necessary, as well as a thorough understanding of design and architectural patterns. Because frameworks take longer to build, those funding the effort should understand that there is a larger up-front cost associated with developing a framework. Experimentation is important as well. Trying different approaches in order to find the most appropriate for a given problem implies a willingness to develop a certain amount of throwaway code.

Because developing a framework is in general significantly more difficult than developing a traditional API, the question of whether or not to use a framework approach must be carefully considered. The extra effort required to develop a framework can be easily re-paid when it is used in more than just a few applications. Frameworks are most suitable in situations where the flow of control will be the same in each of the applications that could potentially use it and there are at least three potential target applications.

Framework architectures should evolve naturally over a period of time [Roberts 1997]. As similar applications are developed, there is often commonality in the flow of control. Once this commonality can be identified, then encapsulating that flow into a framework is possible. Attempting to develop a framework "up front" often results in a very unnatural design that is overly complex. For this reason, the development of a framework should only be considered when one has at least a few existing similar applications and the time to extract the common flow.

Behavior Customization

A framework-based architecture allows the user to specialize the behavior of a framework for specific steps in the process flow. To allow this specialization, a framework defines a component interface for each customizable step. A user of a framework can then develop new components that implement the appropriate interface. The framework then uses the user-provided components to carry out specific steps in the process rather than using the default behavior. This allows a framework to be used in many different contexts. For a framework to operate, there needs to be a way for the framework to communicate with the user-supplied components.

Abstract Interfaces – In many framework-based architectures, the customization is achieved by providing abstract interfaces in the form of base classes. Users are expected to develop derived classes that inherit the appropriate base class. It is in these derived classes where the application specific code is located. By providing abstract base classes, using a framework becomes a fairly straightforward task.

Using inheritance in framework architectures is also known as a "white-box" reuse. This refers to visibility. By providing an abstract base class, a framework makes visible some of its internal details.

Command Pattern – Another way to achieve framework-to-user communication is with the Command Pattern, [Gamma 1995]. This is the approach utilized in the implementation of the Dynamic Terrain Core (DTC) framework. Commands are the object-oriented equivalent of C callbacks. They are first-class objects, so they can have state and be passed between address spaces (unlike pointers to functions) and can take advantage of C++ features such as polymorphism, templates, etc.

With commands, rather than provide an abstract interface that users are required to derive from, a framework would allow users to register user-developed command objects that carry out application-specific tasks. At various steps in the process flow, the framework would invoke the user's command objects, thus specializing the frameworks behavior to meet the application's requirements. This form of composition is known as "black-box" reuse, because no internal details of the objects are visible.

Commands can be used to invoke a wide range of functions including class member functions, static class member functions, and C functions. This allows integration with systems written not only C++, but also in systems developed with languages that can bind with C. [Dewhurst 2000]

DYNAMIC TERRAIN CORE OVERVIEW

A software application framework called the Dynamic Terrain Core (DTC) was developed in order to meet the goals of Phase II. The DTC provides all of the necessary services needed to integrate dynamic terrain capabilities into existing simulation systems including image generators, vehicle dynamics systems, Semi-Automated Force (SAF) systems, and networking systems.

The Modification Service is responsible for the modification of the terrain using a realistic soil model via "Soil Modifiers". This service can be utilized by vehicle dynamics systems for warfighter-in-the-loop simulation, or by SAF systems for automated force terrain interaction.

The Visualization service is responsible for supporting the visualization of the modifications with realistic textures that match the soil type for each area in the database. This service can be utilized by any system that uses a polygonal terrain representation. This includes existing image generation systems and SAF systems, provided that the adding and removing of polygons is supported.

The Distribution Service handles the distribution of terrain changes to other DTC-equipped networked simulators including warfighter-in-the-loop and SAF systems. The distribution can be customized to work with a wide variety of networking protocols.

DYNAMIC TERRAIN CORE ARCHITECTURE

Figure 4 shows the relationship between the various components within the DTC and the client-provided components. The external interfaces to the services provided by the DTC hide much of the complexity of the internal framework architecture that implements the services. This encapsulation allows for future improvements internally without affecting users of the services because the external interfaces will not change. Because the interfaces are simple, using the services is easier. This reduces the cost and time needed for integration.

Hiding complex implementations in such a way is known as the Façade Pattern [Gamma 1995]. The Façade Pattern is used to simplify the use of a complex system by presenting users with a simple, unified interface. Behind the interface are actually many components that work together to accomplish the desired result, but the details of those components and their communications are hidden from the user.

DTC FLOW OF CONTROL

As a framework, the DTC controls the process flow and coordinates the communications between the user-provided components. Figure 5 shows a UML sequence diagram of the flow of communications between the external user-supplied components (indicated with dashed lines) and the DTC components.

Following is a discussion of the data organization and flow of control between the services of the DTC.

Data Representations

Two terrain representations are used within the DTC. The first is a generic triangular representation that corresponds to the terrain polygons found in an IG's visual database. The IG provides this information to the DTC upon the notification of terrain changes. This triangular representation is updated to match those changes.

The second terrain representation is used to efficiently compute the terrain changes caused by soil modifiers as well as distribute those changes to other DTC-equipped simulators. This representation contains soil attribute data and is generated directly from the visual data to ensure correlation. The soil attribute data describes the elevation of the soil along with physical and visual characteristics of the soil at both the surface and at subterranean levels. The generation of soil data can occur either as an offline process or on demand during runtime.

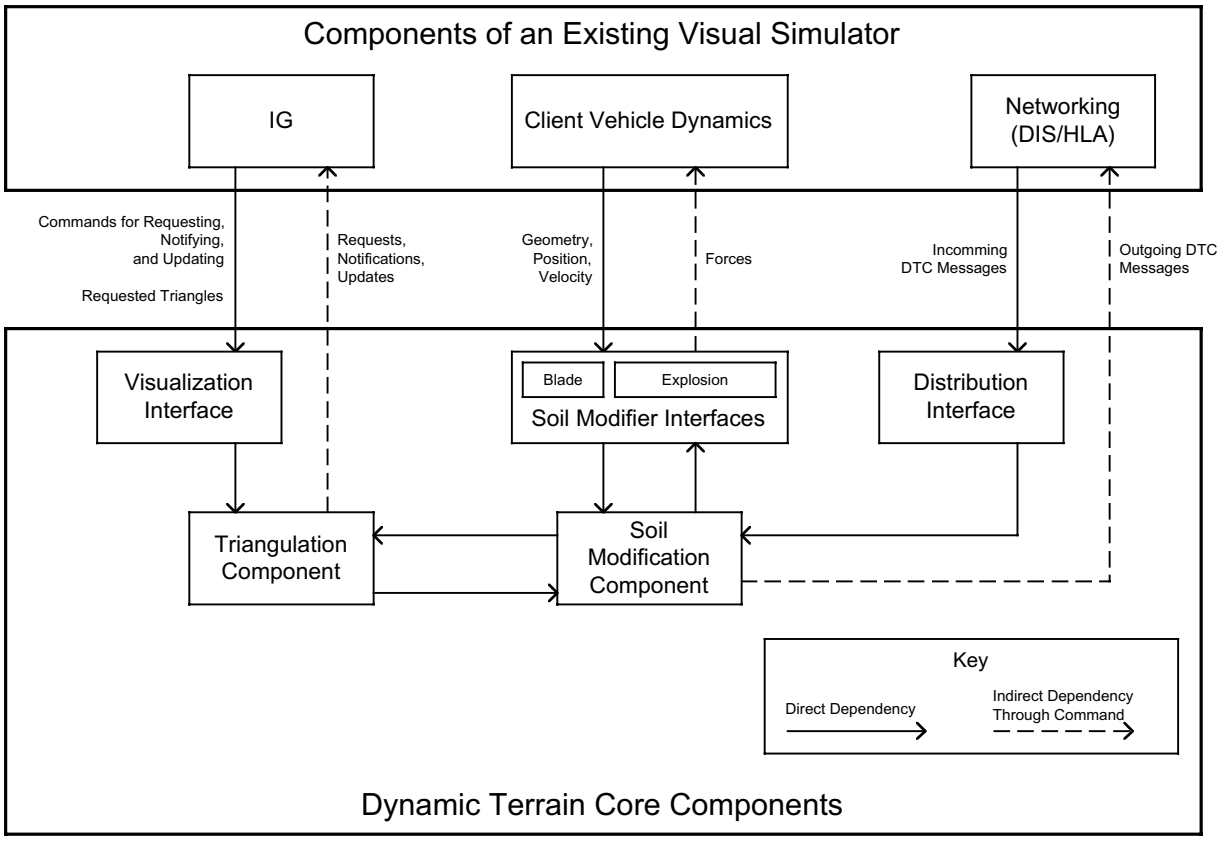

Figure 4 - Dynamic Terrain Core Framework Architecture

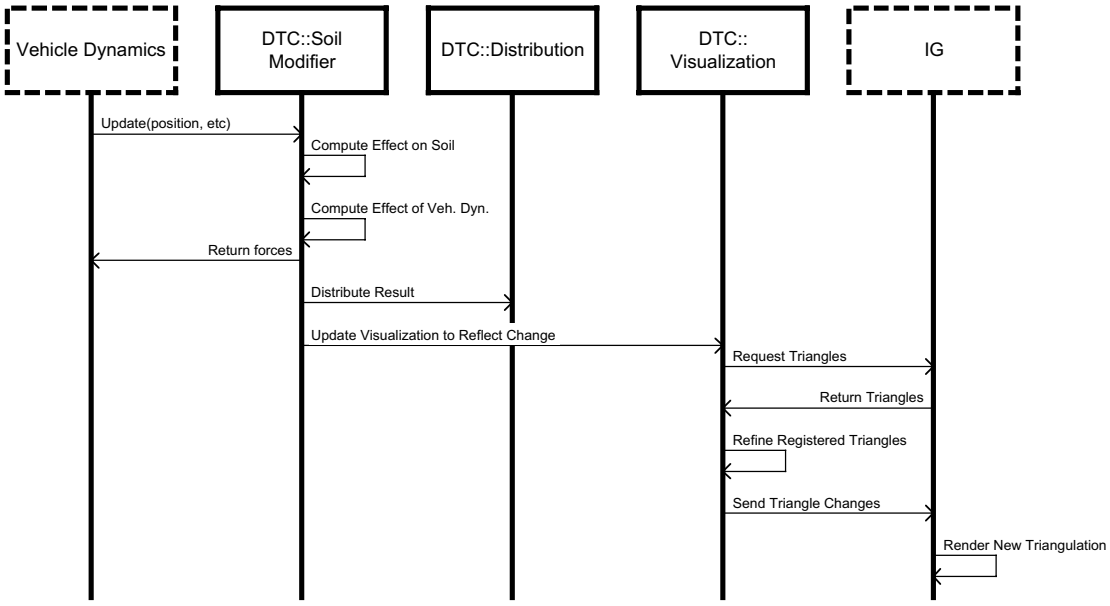

Figure 5 –UML Sequence of DTC Flow

Database Organization

Fundamental to the operation of the DTC is the concept of spatial grouping. Both terrain representations are logically divided into "cells" on a 2-D grid (Figure 6). Spatial grouping allows quick constant-time access to elements in either terrain representation. This is necessary for real-time vehicle interaction and visualization.

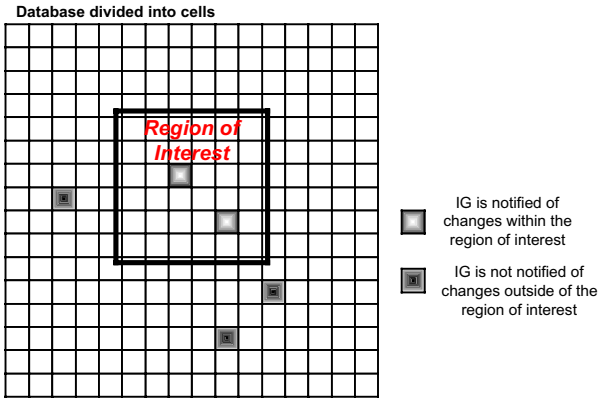

Figure 6 – Database Grouping

All cells are axis-aligned boxes with the same dimensions, so the database information is essentially gridded by the DTC. As updates to the terrain occur (either generated locally via DTC Soil Modifiers or remotely by other networked simulators), the DTC will notify the IG of the cells where changes are occurring. The IG has the option to specify a region of interest so the DTC can filter the notifications sent to the IG. By default, the region of interest is defined as the entire database. The region of interest is useful when very large databases are being used and areas of the database far from the IG's eyepoint are being modified.

Once the IG has been notified about changes in a region of interest, the IG registers the terrain triangles that correspond to changing cells with the DTC. The DTC then adds and removes triangles and vertices to match the surface given constraints for error threshold as well as polygon and time budgets. The constraints allow the IG to control the computing resources used by the DTC. Once a re-triangulation is complete, the DTC updates the IG with the new triangulation.

Efficient Communication: The Mailbox Metaphor

In most existing visual systems, there is little spare processing time available for additional tasks. To reduce the computational burden on the IG, the DTC uses a special method of communication with an IG. This method can be described as a "mailbox metaphor". Each cell in the database is like a house with a mailbox. A postman (the IG) is interested in new mail that needs to be picked up (changes to the terrain). He is only interested in a particular neighborhood (the region of interest). To have mail picked up by the postman, the flag is raised to notify the postman that there is mail (a notification is sent to the IG). The flag is lowered once the postman actually picks the mail up (notification flag cleared).

If the postman does not pick up mail immediately, and more outgoing mail is added, the flag remains raised, so the postman doesn't need more notification that mail is waiting to be picked up. A similar approach ensures that the IG does not receive more notifications than necessary. Only those areas that the IG has shown interest in will continue to receive notifications. The mailbox metaphor is the opposite of the "telemarketer metaphor", where notifications are sent regardless of interest.

Modification

The catalyst of the DTC framework is the Modification Service. To use this service, the existing vehicle dynamics model is modified to update information about the position of objects it contains that could potentially modify the soil (e.g., tires, tracks, plow blades). Changes generated by the Modification Service via Soil Modifiers cause distribution of the changes as well as updates to the visual database. The Soil Modifiers utilize advanced and highly efficient algorithms developed in conjunction with the University of Iowa to allow for realistic soil deformation and the computation of forces that should be applied back to the client vehicle dynamics component.

Distribution

As changes occur as a result of soil modifications, those changes are encapsulated in a message structure that defines the result of the change. The DTC then invokes a client-provided command that sends the message over the network. In this way, the distribution is not directly dependent on any specific networking protocol. It can be adapted to work with DIS, HLA, CORBA, or even low level sockets.

Other DTC-equipped simulators receive the messages and provide them to the DTC. To the DTC, an incoming message from the network is treated in a similar manner to changes made locally via Soil Modifiers.

One question that may arise when discussing distribution is: what is distributed? There are two alternatives. One option is to send a description of the action that caused a change. The other option is to send the resulting state of the terrain due to the change. The DTC sends a description of the resulting state of the terrain. While this approach does increase the demand for network bandwidth, it reduces the computational capability required by each simulator. If only a description of the action were sent, each simulator would need to compute the effect that every other simulator's actions were having on the terrain. In situations with a large number of networked simulators, the burden on each simulator becomes too great [Upton 2000].

Visualization

In addition to distribution, the change needs to be visualized. This visualization, in order to be realistic, must modify the actual surface polygons of an IG's runtime database. This was one of the most challenging aspects of developing the DTC.

In the Phase I implementation of dynamic terrain, the triangulation algorithms were tightly integrated with the runtime database structure. This precluded the use of these algorithms with existing IGs. Each IG has a different method for organizing its runtime database. Some systems provide detailed documentation on the database structure, while others use proprietary approaches. For the DTC to work with existing IGs, a method had to be formulated that would allow for IG-independent manipulation of this data.

To accomplish IG-independence, a generic polygon structure was defined that contains the basic information necessary to support dynamic terrain. This structure includes the identity, vertices, and texture coordinates of a polygon.

When a terrain change occurs due to a local soil modification or an incoming network message, the triangulation component in the DTC is notified. For an update to the IG's runtime database to occur, the DTC needs to be provided with information about the polygons in the visual database at the location of the modification. To get this information, the Visualization Service notifies the IG that a change has occurred in a specific region. The IG responds by supplying polygonal information using the structure described above. The Visualization Service then uses the information describing the soil change to re-triangulate the IG-supplied polygons. Descriptions of the polygonal changes are then sent back to the IG to be incorporated into the IG's runtime visual database.

The update description includes added, deleted, and modified triangle information.

A major advantage to this approach is that there is no direct dependency on how an IG organizes its runtime database, or what file format the database was loaded from. The DTC requests all necessary information at runtime from the various user-supplied components. In this way, dependence on file formats and database structure does not exist, and correlation is automatic. The only requirement is that an IG has the capability to retrieve the terrain polygons from the runtime database in a given area, and to add and remove polygons from the runtime database. A number of scene graphs including SGI's OpenGL Performer support this ability.

CURRENT AND FUTURE WORK

Phase II Plus

Currently, the development of this technology is continuing under a Phase II Plus. In this phase, the DTC is being integrated with an existing simulator developed by Raytheon. The Advance Concept Research Tools (ACRT) desktop simulator is a highly reconfigurable simulation system composed of a number of different computers that run various components of the simulator including a visual system, vehicle dynamics system, sound system, configurable hardware controls, instrumentation display, and many other customizable components.

The ACRT system component software is written in C, so we are developing a thin C adapting layer to allow for the integration of the DTC with C-based systems. Also, a proxy for the DTC is under development that will allow the DTC to be seamlessly hosted on a remote system.

One of the ACRT system computers is a PC running Windows NT 4.0. This computer will host the DTC as well as the existing sound module (which requires relatively little computing power). Having the DTC run on a separate PC has the advantage of not burdening other realtime-critical systems.

The visual component of the system is an SGI Performer-based application running on an O2. Because the Performer scene-graph allows the insertion and removal of polygons at runtime, it is a good candidate for DTC integration. The completion of this effort will provide future "out of the box" support for Performer-based IGs. Additionally, the Phase II Plus effort includes the integration of the DTC with ModSAF 5.0. This effort will be very similar to

integration with a visual system. Essentially, polygons must be inserted and removed from the CTDB (compact terrain database) runtime databases structure. This integration will allow SAF entities to modify the terrain in realtime as well as be affected by the terrain changes made by warfighter-in-the-loop simulators.

Dynamic Objects

In addition to terrain being dynamic in the real world, objects other than the terrain often change as well, especially in a combat situation. The ability to simulate dynamic objects would allow for urban warfare simulation in which buildings and other similar objects would be modified from vehicle interaction, explosions, and other similar influences. Having a more general framework for object forces and their effects on the surrounding environment would allow for future enhancements to the simulation of dynamic objects beyond just the terrain.

Support Off-line Visibility IGs

Some scene management techniques used in IGs for visibility determination are dependent upon a static database structure because visibility relationships are determined off-line. The advantage of off-line visibility determination is that runtime performance is very efficient. In a dynamic environment however, changes to objects could affect visibility. For example, if a munitions blast caused a hole to be formed in the side of a building, portions of the database that would have previously been occluded would become visible. Using offline techniques in such an environment would cause pre-determined information about visibility to become obsolete. This would result in situations with visible objects not being rendered. Further investigation into these topics will help to expand the applicability of the DTC.

CONCLUSION

Given the variations in existing simulation hardware and software architectures, a framework approach to dynamic terrain makes the integration of dynamic terrain capabilities with existing systems possible. While developing a framework is no trivial task, it is more than justified when compared to the alternative approaches: 1) replacing existing simulation systems with proprietary simulation systems designed specifically to allow for dynamic terrain, or 2) re-implement dynamic terrain for each existing system. As the demand for dynamic terrain increases, a flexible framework approach will meet current and future requirements.

REFERENCES

Beedle, M. *Writing Frameworks - What Does It Take?* http://ootips.org/writing-frameworks.html, ootips 1998

Dewhurst, S. *Polymorphic Function Objects*. C/C++ Users Journal, 19(2): 52-60, February 2000

Gamma, E. et al. Design Patterns – Elements of Reusable Object Oriented Programming. Addison-Wesley, 1995

Roberts D. and Johnson R. Patterns for Evolving Frameworks. *Pattern Languages of Program Design* 3. Edited by Robert C. Martin Et al. Addison-Wesley. 1997

Upton, G. et al. Networked Real-time Dynamic Terrain for PCs. IMAGE 2000 Conference Proceedings