Delta3D: A Complete Open Source Game and Simulation Engine for Building Military Training Systems

Perry McDowell
Rudolph Darken
Joe Sullivan
Erik Johnson

MOVES Institute - Naval Postgraduate School
Monterey, CA 93943
[mcdowell, darken, rejohnso, jasullivan]@nps.edu

Delta3D, the open source game and simulation engine built for military training, is continuing to be improved to meet the requirements of the military users. The most recent upgrades, available in versions 1.4 and later, include adding capability for After Action Review, integration with SCORM-compliant learning management systems (LMS's), and distributed interactive simulation (DIS) networking. Additionally, more applications, created by both government users and civilian companies, continue to be built using Delta3D and its expanding capabilities

With these added features, Delta3D has become the engine of choice for several military simulations, including programs of record. The developers and program managers of these programs were attracted by its advanced technical features, its lack of proprietary vendor lock-in and licensing fees, and the ability to quickly produce sophisticated applications using Delta3D.

This paper discusses the current state of Delta3D version 1.4 and how developers and program managers can use Delta3D to quickly and cheaply build complex training systems. It will also briefly touch upon the systems currently being built using Delta3D and how some of these have been proven to work in a training environment. It will also discuss what improvements to the engine will be added in the near future.

Keywords: Game engines, military simulation, game-based training, open source

1. Background

Managers needing visual simulations are caught in a difficult situation. Choosing a modeling tool, image generating system, or software upon which to base their system should not be their primary worry. The only thing they should really care about is how well their final simulation meets its requirements. Unfortunately, under the current simulation business model, this is often a project manager's biggest responsibility, because these proprietary tools consume a huge portion of the project's budget. Too often, this leaves very little funding left to devote to performing needs analysis, building content, or performing verification, validation, and analysis.

***JDMS,* Volume 3, Issue 3, July 2006 Pages 143–154**

These simulation systems are priced as if each is filled with unique features, often times running into the five or even six figures for each application built. However, in reality, a review of most of these simulations reveals that almost all of them have essentially the same features; probably 90% of the functionality each provides is basically the same, with minor differences. The differences between all of them essentially boils down to the most advanced features each provides. However, many of these advanced features are not needed for the vast majority of simulations, especially those designed to run on desktop systems.

Worse, once these choices are made, projects become locked into the proprietary technologies chosen. Normally, managers do not get access to any content or source code created by the contractor. Even

if they did, this content and code would do them little good, because normally that code is written expressly for a specific proprietary engine and cannot be used on any other system. This means that when these managers desire to create a follow-up simulation, they are limited either to using the same developer used for the initial simulation or to paying once again to recreate all the content and code developed originally. Unfortunately, this has rarely led to a lower cost because the original contractor realized the dilemma the manager was trapped in and adjusted bids for follow-on work accordingly. As Doug Whatley, CEO of Breakaway Games—a company that does quite a bit of work building games for the DoD—said, "There's good revenue from owning the IP, but the other thing about it is if they want to do a version 2, they have to come back to you. It guarantees you downstream revenue." [1]

There is another problem using these proprietary systems: there was no way to modify the underlying engine if it did not meet the current needs. Developers can request a feature from the vendor, but such requests rarely result in a modified product timely enough to be useful to the current project. This requires significant developer time and effort to build "work-arounds" to overcome problems with proprietary tools. If the developer had access to the tool's source code, he could easily modify the code to do what he required.

Game designers must overcome similar problems to the simulation manager. Before any development begins, game designers have to choose a proprietary game engine on which to build their games. The licensing fees for these engines normally run from $300,000 up to $1,000,000 for a single application. If the application is successful and a follow-on built, the designer must pay another licensing fee. Additionally, just as in simulations, content is specialized to the engine, so the game designer must either use the same engine or rebuild all code and content, effectively locking them into their original choice.

The U.S. military has been faced with these problems because it is the largest single user of simulations in the world and is fast becoming a major player in the use of games for training. As mentioned above, most of these systems had close to 90% of the same features. In short, they had become commodities. Despite this, system builders are still charging as though they provide a unique product, only available from them. In order to overcome the shortcomings of existing systems for building simulations and games, the MOVES Institute has developed an open source game and simulation engine to allow developers to build systems without facing these difficulties.

2. Design Philosophy

In examining these problems, we came up with a four-part philosophical credo upon which we based building our game and simulation engine:

1) Keep everything open to avoid lock-ins and increase flexibility.
2) Make it modular so we can swap anything out as technologies mature at different rates.
3) Make it multi-genre since we never know what type of application it will have to support next.
4) Build a community (or leverage an existing ones) so the military doesn't have to pay all the bills.

The first of these, "Keep everything open to avoid lock-ins and increase flexibility," addresses two of the problems in the current paradigm. By keeping everything open, no vendor would be able to lock the military into its technology. This would allow follow-on applications to be bid on by multiple companies, with the resulting competition reducing their costs. Additionally, because the tools are open, developers have access to the source code. This means that if the tools don't meet the developers' requirements, the developers can change the tools as needed for their applications without waiting for a vendor to decide to do so.

The second of these tenets, "Make it modular so we can swap anything out as technologies mature at different rates," will allow the engine to be state of the art for a long period of time. Each of the various elements of the game consists of either an open source library or code developed in house. In either case, we have kept the different modules as separate as possible. Therefore, if one of the modules making up Delta3D is surpassed by another open source project and is no longer the "best of breed," it is possible to replace that module with the better one. This can continue with only minor modifications to the Delta3D API, thus allowing the engine to remain current significantly longer than most existing game engines.

The third principle, "Make it multi-genre since we never know what type of application it will to have to support next," is designed to ensure that Delta3D can meet whatever needs the military might have. Just within one service, the Navy, the number of training applications is immense. When he was the commander of the Navy Education and Training Command (NETC) in 2004, Vice Admiral Alfred Harms estimated that he would need approximately 1,500 training games to meet his requirements of performing all individual training within the Navy. Therefore, there is not going to be one genre of games that will be able to meet all those requirements, which are just a small portion of all those in the military. While traditionally game

engines have been built for a single genre, even a single game, that model would not work for the military. By having one engine that can meet all requirements it is easy to standardize the production pipeline and reuse content for multiple applications, thus reducing the cost involved.

The final part of the credo, "Build a community (or leverage existing ones) so the military doesn't have to pay all the bills," is another factor driving us toward an open source solution. The power of open source projects is that the energy of a huge development team can be brought to bear upon problems without actually employing such a large team. By building a well-designed system that people are interested in using for their own applications, they will also add improvements to the original system. Over time, these may add up to have significantly more value than the original system. However, building such a community takes a great deal of time. Leveraging existing open source communities by incorporating current open source projects with large developer bases into the engine creates a built-in group of developers. The advantages this accrues will be discussed more below.

3. Technical Issues and Our Approach

There are several technical issues involved in building an open source game engine. The first is determining how to build the engine. There are several options, such as writing the entire code in-house, choosing an existing open source engine and modifying it to perform all required functionality, or taking several existing open source projects, each of which performs one or more functions needed in the engine, and then hooking these unrelated modules together to produce an engine. Writing the entire engine in-house was rejected as impractical due to time and resource restraints. Several open source game engines were considered, but all had major problems with meeting the requirements of our credo. We determined that modifying each to meet our credo's requirements would require more work and yield an inferior final product. Therefore, our approach to this problem is to use the "best of breed" of previously existing open source software as building blocks for our open source game engine.

This decision has produced many benefits. The first is that we have been able to build a robust engine on a small budget with limited resources. Delta3D itself is a consistent API layer that integrates many existing open source libraries. We proudly claim that we have written approximately 4% (50,000 lines out of 1.2 million total lines in all the libraries that make up Delta3D) of the source code that comprises Delta3D—the rest is existing open source projects. The second benefit is that Delta3D constantly leverages existing open source communities. Our engine is improved not only by "direct" contributors (those who make contributions to the code base of Delta3D), but it is also enhanced by "indirect" contributors (those who contribute to the code base of one of the component projects making up Delta3D). This is a huge advantage, especially to a project in its early stages. The third advantage is that it allows us to maintain Delta3D as being made of the "best of breed," as discussed in the philosophy discussion above. One final advantage is that most open source projects are multi-platform, which means that it was a simple matter to make Delta3D run on multiple platforms. Delta3D has been tested and runs on Windows and Linux. Additionally, it is likely capable of running on operating systems similar to Linux (such as Unix or MAC-OSX), but it has not been tested on these.

The next technical challenges are determining exactly what features to add to the engine, and once this has been done, determining whether an existing open source project could be used to meet the requirement. If multiple open source projects could be used to meet the requirement, then we had to determine which was the best choice to provide that functionality. In certain circumstances, the required functionality does not exist in an open source project and it has to be written from scratch. We have tried to keep Delta3D extremely lean and have begun by adding only those features that are required for the majority of applications. As the use of the engine expands, we (or hopefully, other developers using Delta3D) will add functionality to the engine. As for choosing which projects to use as the modules of Delta3D, we had two criteria: a project's technical merits and its user support base. The rationale for choosing projects upon their merits is obvious, and considering a project's base has allowed Delta3D to gain many "indirect" developers. Additionally, projects with large user bases are more likely to remain current and state of the art than those with only a small base, reducing the likelihood of needing to swap a module.

The initial modules using open source projects, along with the specific projects used for that module, are shown in Figure 1.

One other design issue we felt strongly about and always kept as a factor in design decisions was that Delta3D must be easy to use. We tried to make everything as high level as possible, making it simple for the designer to create objects, have them interact with the other objects in the world, and display the results. For example, it is possible to declare an object that is transformable has physical properties (such as appearance, mass, size, bounding box, animations, etc.) and can be "linked" to other objects. After doing so, designers no longer have to concern themselves

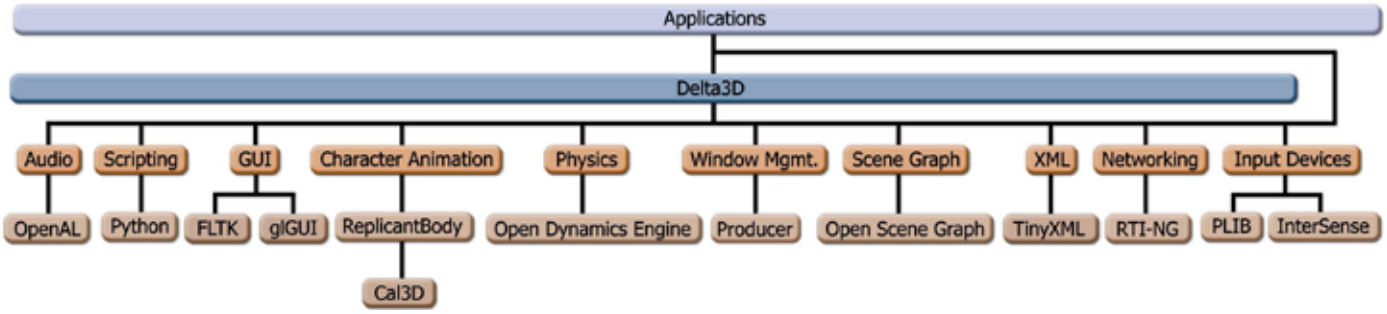

Figure 1. Delta3D's underlying open source libraries

with all those low-level details. They make the object do whatever they want it to, and the engine handles any low-level interactions (positioning, rendering, checking collisions, etc.) that occur. However, if the designers so desire, they can always get to the underlying code if they don't like the way Delta3D handles some of the interactions and they wish to change them. Thus, Delta3D provides the best of both worlds: a simple, easy-to-use API with the ability to completely control all actions.

Another area where we made things as easy as possible for developers is the content creation pipeline. We realize that content creators have their own favorite tools, and we have not imposed any requirements upon file formats that Delta3D will accept. Additionally, we try to make it as easy as possible for content creators to get their work into the engine. For example, we support OSGExp, which is a plug-in for 3DS Studio Max so that it can output files into OpenSceneGraph format. OSGExp has support for geometry, materials, textures, multi-textures, procedural textures, environment maps, cameras, and animations, and has helpers for OSG style levels of detail, billboards, switches, impostors, occluders, node masks, and much more.

Because Delta3D can import many different file formats, the content creator has a wide variety of tools to choose from. In most applications, a blend of open source and commercial content creation tools are used.

4. Description of Delta3D's Libraries

4.1 Rendering

For rendering, Delta3D uses OpenSceneGraph (OSG). OSG is an open source high-performance 3-D graphics toolkit, used by application developers in fields such as visual simulation, games, virtual reality, scientific visualization, and modeling. It is written in Standard C++ and uses OpenGL as its underlying rendering API. It has gained a large following and continues to grow; in a recent poll of visitors to the modsim.org website, OSG was used by more than half of those who responded, as shown in Figure 2 [2]. OSG supports several graphics concepts that greatly improve performance, such as view frustum culling, occlusion culling, small feature culling, level of detail (LOD) nodes, state sorting, vertex arrays, and display lists as part of the core scene graph. It also supports other methods to improve performance, such as customizing the drawing process by implementing *continuous level of detail* (CLOD) meshes atop of the scene graph [3]. Delta3D can use OSG to create realistic scenes with high complexity in real time (> 30 FPS) as shown in Figure 3, a screen shot of a demonstration application built in Delta3D.

Preferred Image Generation Tools (MODSIM, 2005)

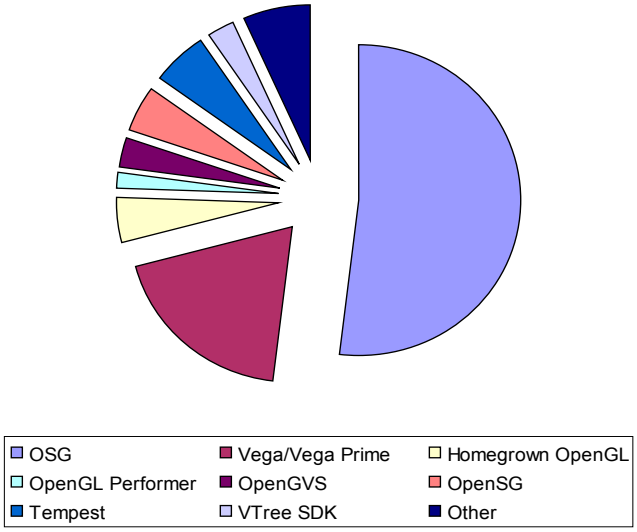

Figure 2. Breakdown of rendering software usage

4.2 Physics

Physics in Delta3D is performed by the Open Dynamics Engine (ODE) library. ODE is a high-performance library for simulating rigid body dynamics. It is fully featured, stable, mature, and platform independent

Figure 3. Screenshot from Delta3D

with an easy to use C/C++ API. It is currently used in several computer games, 3-D authoring tools, and simulation tools. ODE can realistically model several devices/physical phenomena, such as joints, springs, damping devices (e.g., shock absorbers), friction, gears, motors, and collisions. Very advanced rigid body mechanics can be built out of these simulations, providing exceptionally realistic behavior of objects in the games world. ODE uses low-order integration and constraint-based actuators to reduce the amount of time tuning that a developer needs to use to create this realistic behavior. It is particularly useful for simulating vehicles, objects in virtual reality environments, and virtual creatures. [4]

4.3 Audio

Delta3D's audio is handled through the Open Audio Library (OpenAL), which is a software interface to the audio hardware. It resembles the OpenGL API in coding style and conventions and uses a syntax resembling that of OpenGL where applicable. The interface consists of a number of functions that allow a programmer to specify the objects and operations in producing high-quality audio output, specifically multichannel output of 3-D arrangements of sound sources around a listener. Consequently, legacy audio concepts such as panning and left/right channels are not directly supported. OpenAL does include extensions compatible with the IA-SIG 3D Level 1 and Level 2 rendering guidelines to handle sound-source directivity and distance-related attenuation and Doppler effects, as well as environmental effects such as reflection, obstruction, transmission, reverberation.

To the programmer, OpenAL is a set of commands that allow the specification of sound sources and a listener in three dimensions, combined with commands that control how these sound sources are rendered into the output buffer. The effect of OpenAL commands is not guaranteed to be immediate, as there are latencies depending on the implementation, but normally such a latency is not noticeable to the user.[5]

4.4 Character Animation

Delta3D uses the Character Animation Library 3D (Cal3D) to animate characters. Cal3D is a skeletal-based 3-D character animation library written in C++. One nice feature of Cal3D is exporters, which are plug-ins for most popular (both open source and proprietary) 3-D modeling packages. Thus, artists can use their preferred modeling tools to create characters, animations, and textures, and then output them into a format Cal3D can use to control the characters in applications.

The Cal3D C++ library loads exported files, build characters, run animations, and access the data necessary to render them with 3-D graphics. Cal3D can perform animation blending, which allows multiple animations to be executed at the same time with Cal3D blending them together smoothly. This effect allows characters to transition smoothly between different animations, such as walking and running, in any methods to get a wide variety of movement characteristics.

Cal3D provides an automatic level-of-detail control, which improves performance without reducing fidelity by reducing the number of a character's polygons when the character is distant. Also, it is possible to create truly dynamic motion at runtime without the aid of predefined animations. For instance, it is possible to turn a character's head as an object moves past him, rotating the head directly to keep the avatar facing the moving object [6].

In addition to Cal3D, we also use another open source library for character animation. ReplicantBody is a character animation toolkit written in C++, built upon Cal3D and OpenSceneGraph. ReplicantBody is a simple interface for creating and controlling an animated character. It makes a character's movement in the world correspond to that character's feet and makes the avatar follow the ground, making motion appear much more realistic. It also improves the behavior of a character by representing different animation types as actions, and has a manager that keeps track of running actions. This makes it simple to combine actions, i.e., "walk" and "look at" makes a character that continually looks at an object while walking. ReplicantBody is integrated with OpenSceneGraph, which allows it to take advantage of OpenSceneGraph state sorting, greatly improving performance.[7]

4.5 Scripting

The scripting language is one of the most critical factors in allowing advanced behaviors to be added to a game with a minimum of C++ programming on the developers' part. For scripting, Delta3D uses the Python scripting language, which is a portable, interpreted, object-oriented programming language, which has been in development since in 1990. The language has an elegant but not oversimplified syntax, with a small number of powerful high-level data types built in. Developers can extend Python by adding new modules implemented in a compiled language such as C or C++. Such extension modules can define new functions and variables as well as new object types. Python includes classes, a full set of string operations, automated memory management/ garbage collection, and exception handling.

A large number of extension modules have been developed for Python. Some of these are part of the standard library of tools, usable in any Python program (e.g., the math library and regular expressions) and are thus available to developers using Delta3D. Additionally, Delta3D has full binding to connect Python with the C++ code making up Delta3D, which makes it easy for application developers to link their Python and C++ code.[8]

4.6 Additional Functionality

Additionally, there were no open source projects that meet requirements for the following features, so sponsors funded their development either here at NPS or at other companies:

1) Graphical level editor,
2) Advanced terrain/vegetation rendering methods,
3) Advanced environmental features,
4) Particle system editor,
5) Record and playback capability,
6) 3-D model viewer.

One of the most important items contained in Delta3D is the level editor. The level editor, built by the members of the Delta3D team at the BMH operation of Allion Science and Technology, is an easy way for developers to build advanced levels in a graphical manner. The level editor can input all the model types that OSG supports, and the developer can position them in the world, make them move, insert triggers, and incorporate game logic. Level editors such as this are a key part of all professional game engines and make it easy for both professionals and novices to build advanced levels for Delta3D applications.

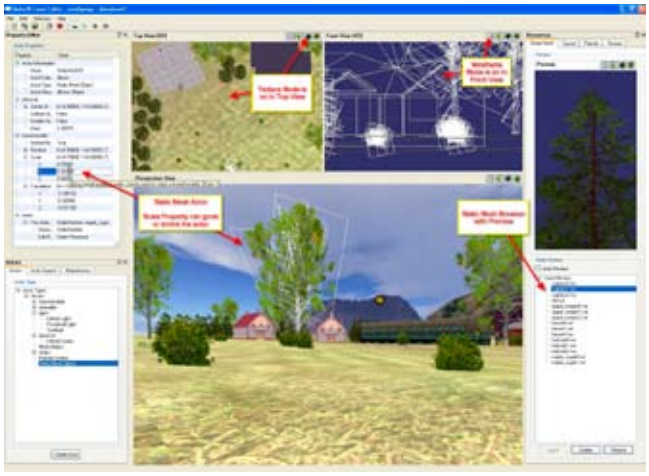

Figure 4. Delta3D level editor

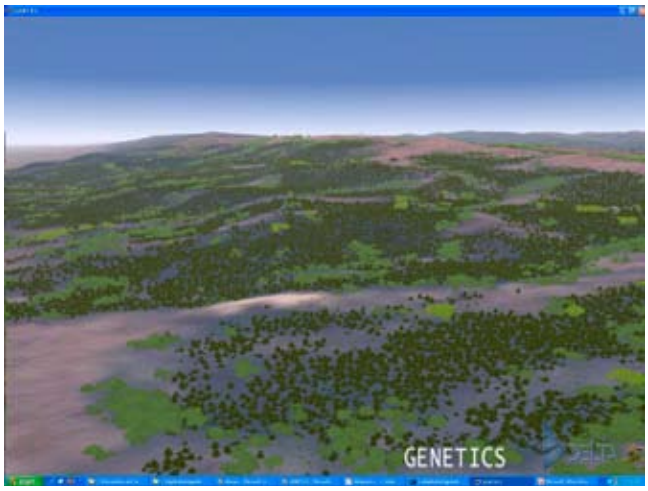

Figure 5. High altitude view of GENETICS terrain and features

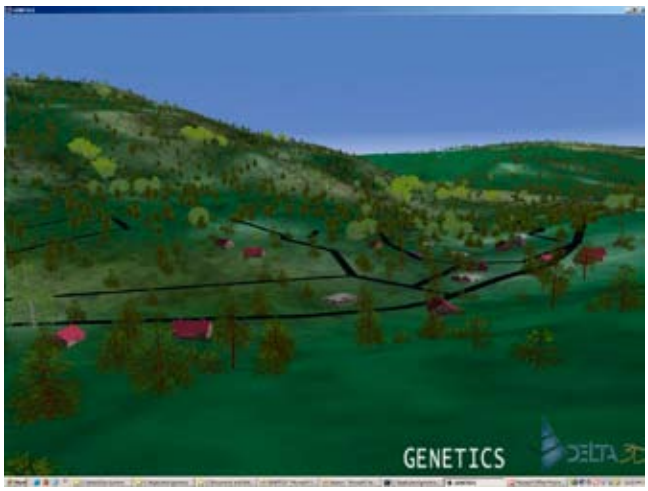

Figure 6. Low altitude view of GENETICS terrain and features

Figure 4 shows an image of the level editor in use.

Delta3D can be used to render extremely realistic terrains with several advantages over current terrain models used in games and flight simulators. Delta3D uses the Generating Enhanced Natural Environments and Terrain for Interactive Combat Simulations (GENETICS) terrain and vegetation engine, created by William Wells, an Air Force Ph.D. student at MOVES. Wells's approach enhances the apparent quality of the given set of terrain elevation data and surface imagery, adds vegetation and man-made objects (such as buildings) that are placed similarly to the arrangement within the actual environment, and generates a plausible synthetic terrain environment where data is missing or incomplete. For further details on how this is accomplished, see Wells and Darken [9].

What this means is that Delta3D offers high-performance rendering of large areas as necessary for traditional jet flight simulators, where the user is operating "high and fast," but also offers the visual cues necessary for flight simulators of aircraft, such as helicopters, which operate closer to the ground, i.e., "low and slow." It is extremely rare for a single engine to be able to provide the required performance and fidelity to perform both tasks well. Figure 5 shows GENETICS terrain from a high altitude, while Figure 6 shows a low altitude terrain, vegetation, and houses placed by the engine.

Another advanced feature of Delta3D is the way we handle environmental features such as the sky, clouds, etc. Once again following our requirement to be as simple as possible, we have built Delta3D to make use of environmental features as high level as possible. Like many engines, Delta3D can use sky boxes to give the atmosphere a realistic appearance. However, developers are limited by the static texture applied to the sky box; the user cannot change weather conditions or time of day. Delta3D can go a step beyond this. By using the sky dome, the built-in ephemeris calculations, and high-level weather controls, the developer can merely input a time and weather conditions (clear, partially cloudy, overcast, etc.) and Delta3D will procedurally generate the clouds and position the sun to match.

One other environmental feature included in Delta3D is procedural clouds. One of the problems with many 3-D games and simulations is that, although their skies appear quite realistic upon first glance, after watching them for some time the user begins to notice that they are unreal, since they never move. To prevent this, Delta3D has two forms of procedural clouds, 3-D clouds and planar clouds, which change over time.

An additional feature contained in Delta3D is a particle system editor. This editor allows developers to use graphical tools to change the properties of a particle system and see the effects immediately in real time. This greatly speeds the development process by eliminating the need to run the application to see the effects of changing a particle system's properties.

Two other features that Delta3D has that many engines do not are a 3-D model viewer and the ability to record and play back scenarios. The model viewer is designed to allow developers to load a model quickly and view it from all angles without having to write an application to do this. The record and playback capability arises out of Delta3D's origins as an engine for training and educational applications. This capability allows both instructors and trainees to go back to a particular moment in a scenario and discuss what was occurring, what the trainee did, and what actions should have been taken.

5. Applications Built Using Delta3D

Prior to the 1.0 release of Delta3D in September 2005, there was already one training application built using it, and several others in development.

The first training application built atop Delta3D is a perfect example of why we feel that Delta3D is necessary to military training. In 2001–2002, David Brannon and Mike Villandre, two Marine Corps students at MOVES, built a trainer for forward observers, those Marines who act as spotter and direct artillery fire [10]. However, this trainer, the Forward Observer Personal Computer Simulator (FOPCSIM), was built atop a commercial development tool for which the MOVES Institute had a development license. However, in order for this application to be shipped and run on PCs to train Marines in the Fleet, runtime licenses would have to be procured for each computer on which the Marines wanted to use it. The cost of these licenses was determined to be too high by the Marine Corps Program Manager for Training Systems (PM-TRASYS), and the system was never deployed to Marines in the Fleet.

In 2004, two other Marine Corps students at MOVES, J. P. McDonough and Mark Strom, decided that the trainer Brannon and Villandre built in 2002 was too valuable to remain unused. They took Brannon and Villandre's code and modified it to run on Delta3D. In addition to not having any licensing fees, access to the engine's source code allowed McDonough and Strom to freely modify the engine to meet their needs, something normally not possible with proprietary solutions. Now, PMTRASYS has made FOPCSIM a program of record and plans to employ it as the trainer of choice for forward observers in the Marine Corps. Additionally, the other three services are planning to use it. Figure 7 contains a screen shot of the current version of

FOPCSIM [11]. This project will be covered further in the section on evaluating performance improvements using Delta3D.

Another application built atop Delta3D at MOVES is a shipboard firefighting application. This was built as a proof of concept of Delta3D's (then called P-51, its development name) ability to serve as the engine of a training application [12]. While never intended to be used as an actual training application, this prototype demonstrated several features that will be useful in building actual training applications, such as recording and playback, a grading system, and feedback to the users as to their performance. A screen shot of the firefighter system is shown in Figure 8.

Another application built using Delta3D is a simulation to train Forward Air Controllers (Airborne) (FAC-A), titled Cleared Hot. In Cleared Hot, the user is the non-flying pilot of an AH-1W helicopter on a mission to direct *close air support* (CAS) in support of an offensive in a desert theater. The user has to complete all the required radio calls, control CAS aircraft, locate the enemy targets, generate a nine-line message, and many other aspects of the FAC-A's job.

Cleared Hot has been incorporated into the Office of Naval Research's Virtual Environments for Training (VIRTE) program. It will be distributed for training to Marine aviators in the future. A screen shot can be seen in Figure 9.

Besides those applications being built at the MOVES Institute, companies are building applications using Delta3D. Applied Visions, a company in New York, has been awarded a Small Business Innovation Research (SBIR) program to build a system to assist launch planners in visualizing Tomahawk missiles flight profiles. Multiple other companies have submitted SBIRs that use Delta3D as the basis of games or visualization systems.

5.1 Evaluation of Delta3D Training Systems

Given that the MOVES Institute built the highly popular recruiting game *America's Army* [13], many people have come to associate MOVES with games and therefore find it unsurprising that we are currently building a game engine. However, at the time America's Army was a bit of an aberration for MOVES—the Institute's forte had always been researching and evaluating training systems. In fact, the genesis of Delta3D was the licensing costs prohibiting the deployment of the original FOPCSIM application. We realized that we, and many others, required an open source engine upon which to build game-based trainers to both measure their training effectiveness and determine effective design paradigms.

Figure 7. FOPCSIM

Figure 8. Delta3D shipboard firefighting prototype

Figure 9. Cleared Hot

While many make extreme claims as to the ability of games to teach people [14], there are others who remain skeptical of these assertions. In order to research the training effectiveness of a given game-base trainer, McDonough and Strom took the FOPCSIM trainer to The Basic School (TBS) in Quantico, Virginia, the initial training for all newly commissioned second lieutenants in the Marine Corps. At TBS, these new Marines are trained in the wide variety of tasks in which a Marine officer is required to be proficient as a platoon commander. McDonough and Strom wanted to determine whether FOPCSIM could replace the current method of training lieutenants in controlling indirect fire.

Currently, lieutenants at TBS are trained in indirect fire using both live and virtual means. However, due to limitations on ammunition and range time, each student is limited to only one live fire mission as part of a team, making it impossible to ingrain the skills via repetition. To overcome this, TBS uses two other systems: the Training Set, Fire Observation (TSFO) system and "lawn darts." TSFO use 35 mm slides to allow students to observe indirect fire, make adjustments to the fall of shot, and see the effects of their adjustments. "Lawn darts" are projectiles fired from an 81 mm mortar; they allow the student to see the entire range and observe the operation of the team operating the mortar and the way in which they respond to the student's call to adjust fire.

McDonough and Strom hypothesized that lieutenants whose TSFO training was replaced with FOPCSIM-based training would perform better than those who trained using the current method. They divided a TBS class into two groups: 166 of the students were trained using the TSFO, while 61 received two hours of supervised time on FOPCSIM and were allowed to use it as much as desired during their off hours. While the best measure of training effectiveness for this experiment would have been having seasoned instructors grade each student on their ability to perform call for fire in the field, this was impractical due to ammunition and range constraints. Instead, the results of the portion of the Supporting Arms Exam (SAE) dealing with call for fire were used to determine each student's ability.

The results of the experiment are shown in Table 1. They showed that the students trained using FOCPSIM scored significantly better ($p < .05$) than those who used the current training method, TSFO. Interestingly, while approximately half of the students trained with FOPCSIM took advantage of the opportunity to use it outside of the two hours of observed class time, those who used it more did not score better than those who did not. In fact, the opposite is true; while both of the groups using FOPCSIM scored better than the control group, the group that did not use FOPCSIM outside of class scored better than those who did (86.91 versus 83.84). McDonough and Strom attribute this to the fact that those who felt they were having difficulty with the subject were more likely to use it outside of class than those who felt proficient after the two-hour class.

Table 1. Supporting Arms Exam results; over-all score (from McDonough [10])

Group	N	Mean	Standard Deviation	Std. Error Mean
FOPCSIM	61	85.348	10.039	1.285
TSFO	166	82.096	9.967	0.773

Another important result for this experiment is that FOPCSIM acted as an outstanding predictor of failure on the SAE. Figure 9 shows the correlation between the students' scores assigned by FOPCSIM's scoring system and passing the SAE. No student who scored below 85 on FOPCSIM failed the SAE, while those who scored below 85 had over a 20% failure rate (4 of 19). This is extremely important, since instructors can use the results of the students' performance on FOPCSIM to target those who are in danger of failing for extra instruction prior to the exam in the hopes of bringing them to a passing knowledge level. More information on McDonough and Strom's experiment can be found in [10].

McDonough and Strom's experiment shows how Delta3D can be used to build experimental training systems, which can then be used to investigate the ability of game-based systems to train personnel in tasks. We feel that this is an extremely important ability, and one that will likely be one of the most important uses here at the MOVES Institute. In the future, we plan to continue building such systems and evaluating game-based training in a wide variety of tasks. Additionally, we plan to use Delta3D to investigate what features of game-based training are most important in creating an effective training application.

6. After Action Review and LMS Interaction

The United States military is in the process of changing the way all training is done. One key component of this transformation is the development of the integrated learning environments (ILE's). ILE uses multiple methods of instructional delivery to meet the military's myriad training requirements, attempting to match the best method to the combination of trainee and subject. To tie the multiple methods together, the Advanced Distributed Learning (ADL) initiative has mandated

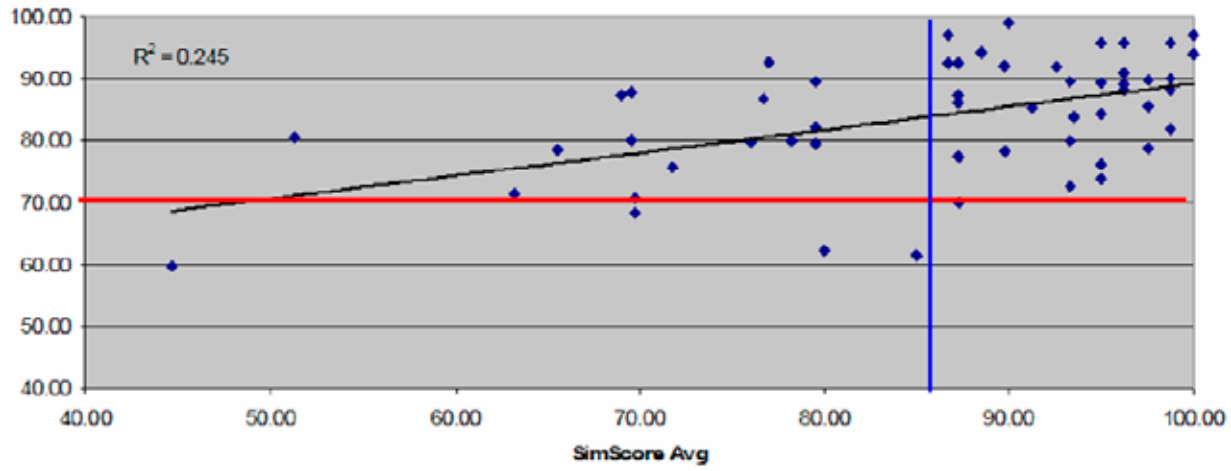

Figure 10. Supporting arms exam score versus SimScore average (from McDonough [10])

that all training materials meet the guidelines promulgated in the Sharable Content Object Reuse Model (SCORM) for all learning materials to be used within the ILE.

However, SCORM has no guidelines to integrate the power of interactive simulations into the ILE. NETC's Experimentation Lab, located at the NAVAIR Training Systems Division in Orlando, Florida, is researching the best methods to do this. One of the areas of research is the tracking of learning objectives between a learning management system (LMS) and a gaming/simulation engine for reporting a user's results using the simulation to the LMS.

As part of this research, NETC contracted with the BMH and Engineering and Computer Simulations (ECS) to develop technology and techniques that would allow a Delta3D application to be packaged and deployed for use within an LMS. An LMS typically consists of server-based components that deliver training content to students via web browsers, and acts as a database providing centralized management of the content as well as student information, such as modules completed and performance data. This project developed a stand-alone suite of tools for packaging a game-based Delta3D application so that it can be downloaded, installed, and launched from a web server with or without support for an LMS.

Additionally, ECS developed a method for the simulation to report back to a SCORM-conformant LMS in real time as the trainee meets performance objectives. This allows the LMS to document the trainee's competence, or lack thereof, in certain areas.

7. Artificial Intelligence

Recently, several new improvements have been added to Delta3D. The most significant of these is the addition of an artificial intelligence (AI) framework to the engine. It is based upon Jeff Orkin's planning architecture [15] used in the game F.E.A.R., which received rave reviews for its AI. The planning system allows programmers to define "plans" and let the computer determine how to complete the plans. It allows the programmer to create complex AI behavior without having to generate an incredible number of states, as in a traditional finite state machine. Additionally, as new objects and behaviors are introduced into the game, the programmer is not required to modify every state and create new states as in a finite state machine. This is a significant improvement over traditional game AI and should allow building applications of increasing scope and complexity in Delta3D.

8. Impact

Delta3D has the capacity to significantly change the way that serious gaming and military simulations are done. The current paradigm of the military paying multiple times for the same commodity can finally be ended. Additionally, the days when the initial developer was the only contractor who could expand or modify a system—and could charge whatever exorbitant fee desired because the application was tied to that contractor's proprietary tools—are over. The military will be able to pay whichever contractor can provide the best value in upgrading and maintaining training applications.

Another area where Delta3D will make a huge difference is the academic arena. For quite a while, traditional computer graphics and virtual worlds classes have needed a simple API for students to create advanced applications for hands-on experience and to experiment with new ways of doing things. With the huge increase in interest in games as a career

and a business, many schools are beginning to offer classes and degrees in gaming, and the need is even more pressing there. While some of these are well funded by industry, most have limited resources, especially community colleges, where a large portion of these programs are being created. Delta3D offers an outstanding choice for academic institutions looking for an API upon which to build class projects, theses, and other such applications.

Additionally, Delta3D makes a great platform for anyone who wishes to build either a game or simulation without a big budget, such as small companies or people desiring to build a game as a demo to help them get into the gaming industry.

9. References

[1] Sheffield, B. "Breaking the Waves: Doug Whatley and BreakAway Games Get Serious." *Game Developer Mag* (February 2005): 25–26.

[2] MODSIM.org [Homepage on the Internet]. Updated March 15, 2005. Cited March 17, 2005. Available from: http://www.modsim.org/devrim_extras/poll_piechart_scenegraph2005.png

[3] OpenSceneGraph [Homepage on the Internet]. Cited March 17, 2005. Available from: http://www.openscenegraph.org/

[4] Open Dynamics Engine [Homepage on Internet]. Cited March 18, 2005. Available from: http://ode.org/

[5] Open Audio Library [Specifications Page on Internet]. Cited March 18, 2005. Available from: http://www.openal.org/oalspecs-specs/x44.html

[6] Character Animation Library 3D [FAQ Page on Internet]. Cited March 18, 2005. Available from: http://cal3d.sourceforge.net/docs/api/html/cal3dfaq.html

[7] ReplicantBody [Homepage on Internet]. Cited April 8, 2005. Available from: http://www.vrlab.umu.se/research/replicantbody/#doc

[8] Python Software Foundation [Introduction Page on Internet]. Cited March 18, 2005. Available from: http://www.python.org/doc/Introduction.html

[9] Wells W.D., and C.J. Darken. "Generating Enhanced Natural Environments and Terrain for Interactive Combat Simulations (GENETICS)." In *Image 2005: Proceedings of the IMAGE Conference.* Tempe, AZ, The IMAGE Society, 2005.

[10] Brannon, D., and M. Villandre. "The Forward Observer Personal Computer Simulator (FOPCSIM)." Master's thesis. Monterey, CA, Naval Postgraduate School, 2002.

[11] McDonough, J., and M. Strom. "The Forward Observer Personal Computer Simulator 2 (FOPCSIM)2." Master's thesis. Monterey, CA, Naval Postgraduate School, 2005.

[12] McDowell, P.L., and R.P. Darken. "Using Open Source Game Engines to Build Compelling Training Simulations." In *Proceedings of Interservice/ Industry Training, Simulation and Education Conference.* Orlando, FL, 2004. Arlington, VA: National Training Systems Association, 2004.

[13] Zyda, M., A. Mayberry, J. McCree, and M. Davis. "From Viz-Sim to VR to Games: How We Built a Hit Game-based Simulation." In *Organizational Simulation: From Modeling & Simulation to Games & Entertainment.* edited by W. Rouse, and K. Boff, 25–32. New York: Wiley, 2004.

[14] Prensky, M. *Digital Game Based Learning.* New York: McGraw-Hill, 2001.

[15] Orkin, J. "Three States and a Plan." In *Proceedings of Game Developers' Conference (GDC) 2006.* San Jose, CA: CMP Media, 2006.

Acknowledgements

We would like to thank our sponsors, the NETC Learning Strategies Division, the Joint Force Command Joint Warfighting Center for funding to incorporate Delta3D into the Joint National Training Capability, and the Naval Modeling and Simulation Office.

Author Biographies

Perry McDowell *is a former Naval Nuclear Power Surface Warfare Officer. He has been on the faculty of the Naval Postgraduate School since 2000, where he teaches computer science, does research in virtual environments and training for the Modeling, Virtual Environments, and Simulations (MOVES) Institute, and serves as Executive Director for the Delta3D Open Source Game and Simulation Engine. He is currently conducting research for his Ph.D. He graduated with a B.S. in naval architecture from the U.S. Naval Academy in 1988, and an M.S. in computer science (with honors) from the Naval Postgraduate School in 1995.*

Rudolph Darken *is an Associate Professor of Computer Science and the Director of the Modeling, Virtual Environments, and Simulation (MOVES) Institute at the Naval Postgraduate School in Monterey, California. He also directs the Laboratory for Simulation and Training and modeling and simulation efforts for the Center for Homeland Defense and Security. His research has been primarily focused on human factors and training using virtual environments and computer gaming media with emphasis on navigation and wayfinding in large-scale virtual worlds. He is a Senior Editor of* PRESENCE Journal, *the MIT Press journal of teleoperators and virtual environments. He received his B.S. in computer science engineering from the University of Illinois at Chicago in 1990 and his M.S. and D.Sc. degrees in computer science from The George Washington University in 1993 and 1995, respectively.*

Erik Johnson *has been Lead Engineer for the Human Performance Engineering and Game-Based Simulation group Research Associate at the Modeling, Virtual Environments, and Simulations (MOVES) Institute since 2001. Previously, he was a key software engineer for Boeing Helicopters in Mesa, Arizona, where he helped design and develop real-time graphical simulations for future rotorcraft designs. Mr. Johnson is one of the primary founders of the Delta3D Open Source Game and Simulation Engine and is actively managing the engineering efforts. He graduated from the Embry-Riddle Aeronautical University in 1995 with a B.S. in aviation computer science.*

Commander Joseph Sullivan *is an SH-60F pilot who has performed several sea tours assigned to both squadrons and ships. His is currently an instructor at the Naval Postgraduate School in the Department of Computer Science and a member of the Modeling, Virtual Environments, and Simulations (MOVES) Institute. CDR Sullivan graduated from Catholic University of America in 1986 with a B.S. in computer science and from the Naval Postgraduate School with a M.S. in computer science. He is currently pursuing a Ph.D. in modeling and simulation.*