2005

Applying Gaming Technology to Tomahawk Mission Planning and Training

Doris, Ken

Monterey, California: Naval Postgraduate School.

Applying Gaming Technology to Tomahawk Mission Planning and Training

Ken Doris
Mark Larkin
Applied Visions, Inc.
Northport, NY 11768
631-754-4920
kend@avi.com
markl@avi.com

David Silvia
Naval Undersea Warfare Center DivNpt
Newport, RI 02841
401-832-2869
silviada@npt.nuwc.navy.mil

Perry McDowell
The MOVES Institute
Monterey, CA 93943
831-656-7591
mcdowell@nps.edu

Keywords:
Computer Game Engine, 3D Visualization, Tomahawk Missile, Vehicle Movement Prediction

ABSTRACT: *Over the past decade the computer gaming industry has not only generated its own multi-billion dollar section of the entertainment industry, but it has also made significant inroads into the military market, especially in training and simulation, starting with Marine Doom and continuing up to today's Full Spectrum Command and America's Army.*

This paper describes a Navy-funded research project that uses gaming technology for not only training, but also as an operational decision aid for the Tactical Tomahawk Weapon Control System (TTWCS). The research is aimed at adapting game engine technology to predict and simulate the motion of ground target vehicles (e.g. SCUD Launchers) through their local terrain over a given period of time, then use the associated rendering capabilities to provide realistic 3D views.

The paper presents an overview of the TTWCS mission and how it will benefit from specific advances in gaming technology, especially in the areas of artificial intelligence, path finding, and physics. It discusses the current state of the project using existing commercial gaming technology and the future plans for adapting and expanding the open source game engine technology of the Delta3D project underway at the MOVES Institute at the Naval Postgraduate School.

1. Introduction

The Tactical Tomahawk missile is the latest in the evolution of the Tomahawk weapon system. It adds the capability to reprogram the missile in-flight to either strike preprogrammed alternate targets or to redirect the missile to newly found targets of opportunity. The missile is also able to loiter over a given area, either using its on-board camera to assess the target, or operating in a stand-off mode, waiting for a target to enter a

kill window, perhaps some distance away. Figure 1 below illustrates some of these capabilities

The ability to reprogram the missile in-flight is provided by the Tactical Tomahawk Weapon Control System (TTWCS). The success of TTWCS will depend on how well a weapons officer can decide *which* missile should be assigned to *what* target and *when* it should carry out its attack. The problem is a complicated one, and typically includes multiple targets and threats, with complex time/distance relationships. The need to provide the operator with a method to quickly and easily visualize the possible future movements of those targets and threats is clear. How to design and implement such a system was the question asked by the Navy in an SBIR topic in 2004. Applied Visions (AVI) was one of three awardees of a Phase I contract and was recently selected as the finalist to continue into Phase II. This paper describes the progress made under that Phase I contract as well as the plans for Phase II.

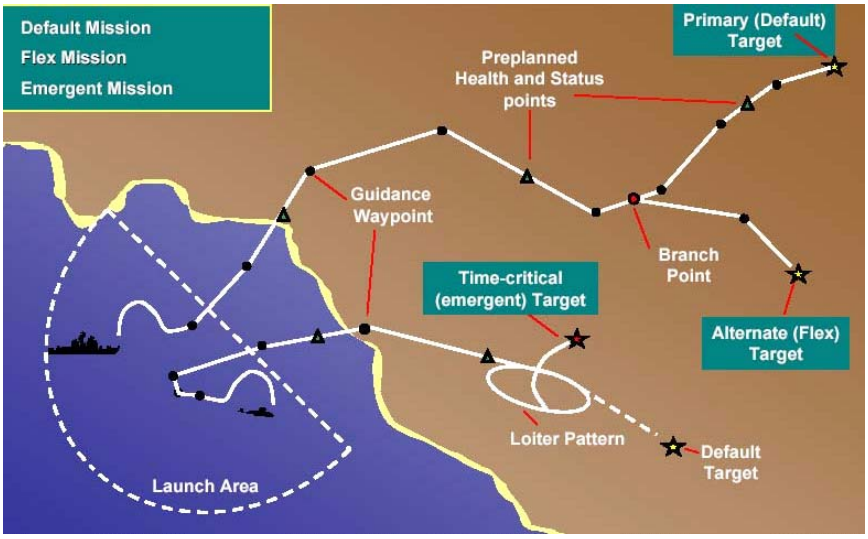


Figure 1 – Tactical Tomahawk Missions

2. Motivation

The Tomahawk Land Attack Missile (TLAM) is a long range, subsonic cruise missile, launched from U. S. Navy surface ships and U.S. Navy and Royal Navy submarines. Tomahawk missiles, used for land attack warfare, are designed to fly at extremely low altitudes at high subsonic speeds and are piloted over an evasive route by several mission-tailored guidance systems. There are two variants of the Tomahawk Missile currently deployed, the Block III and the Block IV. Both feature an Inertial Navigation System (INS) aided by Terrain Contour Matching (TERCOM) for missile navigation and a Digital Scene Matching Area Correlation (DSMAC) and Global Positioning Satellite (GPS) System, which are coupled to the guidance systems to provide precision navigation. In addition the Block IV, or Tactical Tomahawk, features the ability to reprogram the missile in-flight and strike alternate targets at any Global Positioning System (GPS) coordinates. It also has the ability to loiter over a target area and provide target battle damage assessment using its on-board camera.

The Tomahawk missile has become the weapon of choice for the U.S. Department of Defense because of its long range, lethality, and extreme accuracy. They are used against high-priority, long-dwell targets whose priority does not change during the missile's transit time [1].

However, the Tomahawk has limited effectiveness against short-dwell or time-critical targets, and has never been used against mobile, high-value targets such as mobile missile launchers. These targets present special challenges for the weapon system because the missile cannot be retargeted quickly. Furthermore, the Tomahawk missile has limited endurance, increasing the likelihood that it will run out of fuel before new target solutions can be determined [2].

With the advent of the Tactical Tomahawk, with its ability to be queried and controlled, new technologies must be developed and optimized in order to increase weapon effectiveness within an ever changing battlespace. One area that must be addressed is Situational Awareness. Situational Awareness supports the operator by providing the correct level of information to make decisions in a timely manner. The operator needs to understand the battlespace relative to land attack from a sea-based platform, in order to optimize missions and reduce missile to target times. For the TTWCS operator it is as important to understand future battlespace, as well as the current. Specifically, the operator needs to predict the movement of time-critical/ mobile targets, reduce collateral damage, and understand the effects of weather within a geographic area. This level of understanding is only possible through the development of tools that allow the operator to visualize current and future events. It is for this purpose that this SBIR topic was proposed.

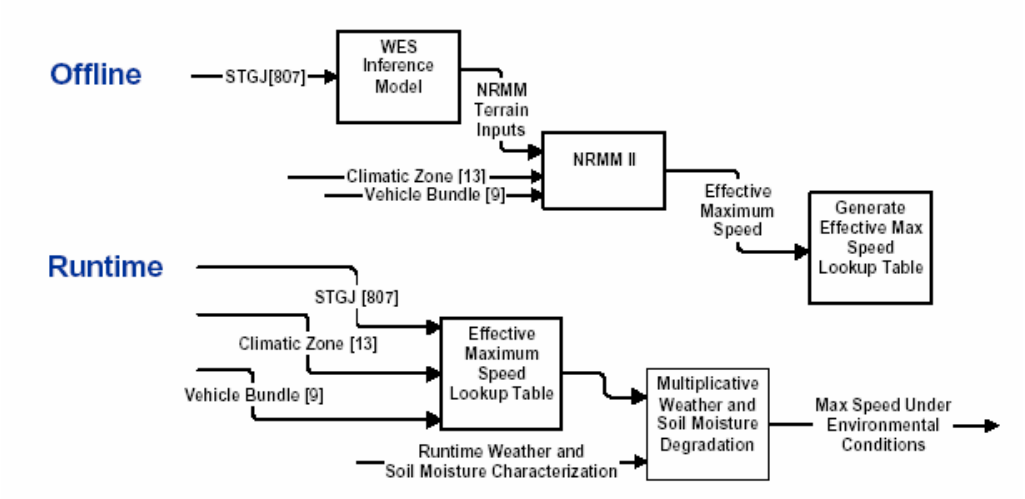


Figure 2 – WARSIM Mobility Model

3. Background

A number of existing military systems are used to predict or simulate ground vehicle movement. We first analyzed several of these before determining that using gaming technology was the best solution.

Adapting Existing Military Systems

One approach to providing these capabilities is to start with existing DoD mobility models and adapt them to the TTWCS. For example, one of the most widely used models is the NATO Reference Mobility Model (NRMM), originally developed in the 1970's by the US Army's Waterways Experiment Station (WES).

The NRMM is a set of equations and algorithms that predict a particular vehicle's performance in a prescribed terrain based on vehicle characteristics and terrain properties. The main prediction module considers vehicle, terrain, and vehicle/terrain-independent scenario data such as weather conditions, to determine the maximum possible speed at which the vehicle can operate. The NRMM is primarily used during a vehicle's design stage to predict the ability of that vehicle to traverse a given type of terrain. Adapting it to the TTWCS mission would require significant effort, as it is a computationally intense, engineering-level simulation. As a result it runs considerably slower than real-time. The TTWCS operator will need to analyze multiple paths quickly, thus any system employed must be capable of running faster than real time.

Another potential source of a suitable mobility model is the military training community. Mobility models in trainers are typically employed to simulate the vehicles of opposing forces. This class of models can range from basic doctrine-driven approximations to sophisticated systems that incorporate engineering data for soil and vehicle interaction. These basic systems can be eliminated from consideration as they rely on simple generic rules for both vehicles and terrain, and are aimed at company level or higher resolution. The more sophisticated systems offer much higher fidelity, such as that used in the Army's WARSIM project [3]:

> *"WARSIM (Warfighting Simulation) is a simulation developed for the purpose of training U.S. Army commanders and their staffs from a battalion through a theater level war. The level of fidelity in modeling ground vehicle mobility considers platform-level issues; however a large amount of behavior will be aggregated to the platoon level."*

The WARSIM mobility model uses NRMM as an offline component, as illustrated in Figure 2. This model makes heavy use of lookup tables to characterize over 200 towed, wheeled and tracked ground vehicles, while terrain is handled with over 800 terrain codes. Although the WARSIM model currently only runs in real-time or slower, it is possible that a version to run at speeds needed for the TTWCS problem could be developed. One difficulty in modifying this system is the fact that the application has already been optimized for speed via the use of table lookups (vs. running equations). A second, and perhaps more significant problem in that effort, would be the difficulty of working with the WARSIM code base, which is relatively old (the project began in the mid 90's).

We next began looking at the commercial world for mobility-related technology and found the most striking examples in the entertainment industry – in computer games.

4. Why use Gaming Technology?

The problem of predicting vehicle movement within the TTWCS context can be broken down logically into three essential parts: *prior history, current state,* and *future goals*. By analyzing and combining these elements, it is possible to determine the *probable movement* of the vehicle. In Phase I we began attacking this problem by dividing it into the following technology areas:

- ***Artificial Intelligence*** – "memory" of prior events, "sensing" current status and events, and decision making, all resulting in evaluation of holding position or moving to a new destination.
- ***Pathfinding*** – given a decision to move to a new destination, what routes would the ground vehicle take and what are the relative benefits and risks of each path?
- ***Vehicle Physics*** – how quickly can the vehicle move along each route to reach the potential destinations?
- ***Visualization*** – provide the operator with a user interface (UI) that allows intuitive interaction and rapidly increases situational awareness.

All four of these elements are found in modern game engines. Computer games now exist that handle both tracked and wheeled vehicles, running on and off road. The proliferation of these games is astonishing, with literally hundreds of titles currently available for the PC platform alone. Driven by a huge mass market, with R&D costs spread over a vast revenue base, game engines offer a "best in breed" technology that continually improves with each year at a breathtaking rate.

The following sections describe our work in each of the four technology areas.

5. Artificial Intelligence (AI)

Our research looks into adapting the computer-controlled forces (often referred to as *Non-Player Characters*, or NPC's) that play a large part in most modern games. The technology has rapidly evolved to the point that, in some of the latest games, the NPCs exhibit behaviors that in many ways appear to be sentient. We are seeking to adapt this cutting-edge technology to predict the future actions of hostile ground vehicles.

In looking at the architecture of each of the candidate game engines we found that while the more established functions such as audio and physics were implemented as true subsystems, it is difficult to separate out the AI code from the rest of the game engine logic. This reflects the fact that, over the last decade, companies such as Havok and Novodex (now Ageia) have produced and marketed separate products for those technologies. AI is a relative newcomer to this paradigm, and the emergence of the technology as a separate product has only recently occurred, with what are termed "AI middleware" products. We researched the market's most recent products, and arrived at the following list of candidates:

- AI.implant, from BioGraphic Technologies
- DirectIA, by Mathematiques Appliquees
- RenderWare AI Middleware (RWAI), by Kynogon
- SimBionic, by Stottler Henke

By reviewing the literature on each of these products we were able to create a comparison chart, as shown in Figure 3.

While all of these candidates appear viable for the TTWCS application, we narrowed our choices down to *AI.implant* and *SimBionic* based on attributes which we felt were critical to the TTWCS application: high-level editors combined with user-developed behaviors. We finally selected *AI.implant* as our Phase I choice based on the availability of a free development license combined with good documentation and an active user community.

AI.implant's development environment (AI.DE) includes an editor that allows creation and modification of NPCs, including a full definition of the "brain" of the NPC. This brain consists of the following elements:

- Knowledge of one's own capabilities and performance parameters
- Knowledge of prior events (world and self history)
- Knowledge of current state
    - Internal status, such as fuel, speed, etc.
    - External status, such as proximity to threats, gained through the use of user-defined sensors

- Decision Logic – given the current state and prior history, what action, if any, to take next.

All of these elements are completely customizable without writing any code, thus putting the design of the "brain" directly into the hands of an *Analyst* without needing the support of a *Programmer.* This type of AI architecture can be tremendously useful in the context of TTWCS. Figure 4 contains a screenshot taken from the *AI.mplant* development environment showing the development of an NPC,

The current version of *AI.implant* allows two types of decision logic: *Decision Trees* and *Finite State Machines (FSMs).* In Phase I we experimented with Finite State Machines as the primary decision logic, as they are more efficient at handling large numbers of input variable and output states. While this approach was certainly sufficient for the Phase I Prototype, we fully realize that other options need to be explored as we move forward. For example, an FSM might not be the best solution since it expects *crisp data* as inputs, and produces Boolean results for the possible output states with only one of *n* states declared *true*. While having many possible states is one way to improve the FSM flexibility, it is costly in terms of both complexity and resource utilization.

In Phase II of the SBIR we will explore the use of *Fuzzy State Machine* (*FuSM*) technology. In a FuSM the logic allows *degrees* of *true*/*false* for each output state through the use of *membership functions* which map input variables into a degree of membership in a fuzzy set between 0 and 1. A *FuSM* doesn't produce a single answer, but rather shows multiple possible answers, each with a weighting factor. This type of output may be better suited to the TTWCS application.

6. Pathfinding

In the previous section we discussed how we are adapting gaming AI to predict the next most likely actions of the hostile ground vehicles. In many cases this will result in the decision to move to a new location. How the vehicle might traverse the distance from its current position to that location is the next problem to be solved.

A hostile ground vehicle at a given location will traverse to a new location in accordance with a number of dynamic factors. It will avoid exposure to threats, follow the most efficient path, etc. In Phase I our goal was to experiment with how easily the pathfinding logic in game engines could be adapted to the TTWCS application.

Starting with our initial list of approximately a dozen candidate game engines, we narrowed the choice to *UnReal* and *Torque* since we had access to source code for both and were reasonably familiar with their code base. The versions of each engine we had in our possession employ relatively rudimentary AI pathfinding based on pre-scripted waypoints. We believe that entering waypoints into each terrain map is both error-prone and overly time consuming, so we decided to modify the *Torque* engine to include a dynamic pathfinding capability based on the "*A**" (pronounced "A-star") algorithm. We chose A* for its efficiency: it is widely used in modern games, and is the subject of continual analysis and improvement by the gaming community [4].

Given a starting point and a destination, A* dynamically builds paths by evaluating the "cost" of each possible route section, with an overall goal of generating the "least cost" path. For Phase I our only cost function was terrain slope. Using the slope of each terrain tile, A* will build a path that follows the flattest (easiest) path between the two points.

	Product			
Feature	**AI.implant**	**DirectIA**	**RWAI**	**SimBionic**
User Decision Support	Binary decision trees	Motivated decision graphs	Finite state machines, neural networks	Finite state machines
Other Services	Auto path generation, pathfinding	Pathfinding	Graphics and physics in other modules, auto path generation, pathfinding	Inter-agent communication
Behavior Support	Pre-packaged behaviors	Templated behavior scripts	Pre-packaged behaviors	User developed behaviors
Engine Source Code Availability	Some	No	Yes	Some
Extensibility	User-developed behaviors	User-developed scripts, callback functions	User-developed behaviors, callback functions	User developed behaviors, callback functions
Production Tools	Maya/3ds max plug-ins	Script templates, tuning GUI	AI skeleton code, XML configuration	Visual editor, visual debugger

Figure 3 – AI Middleware Product Comparison

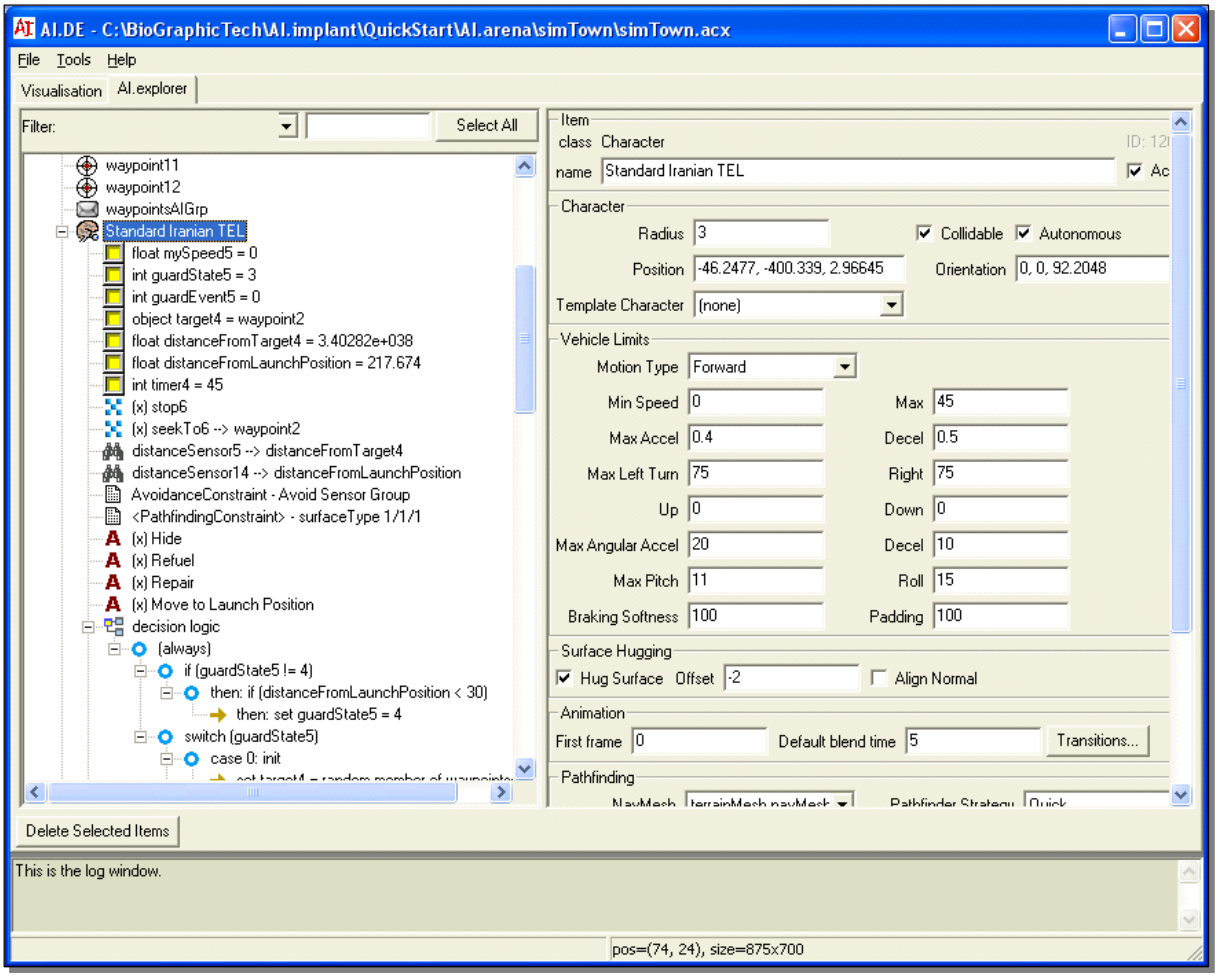
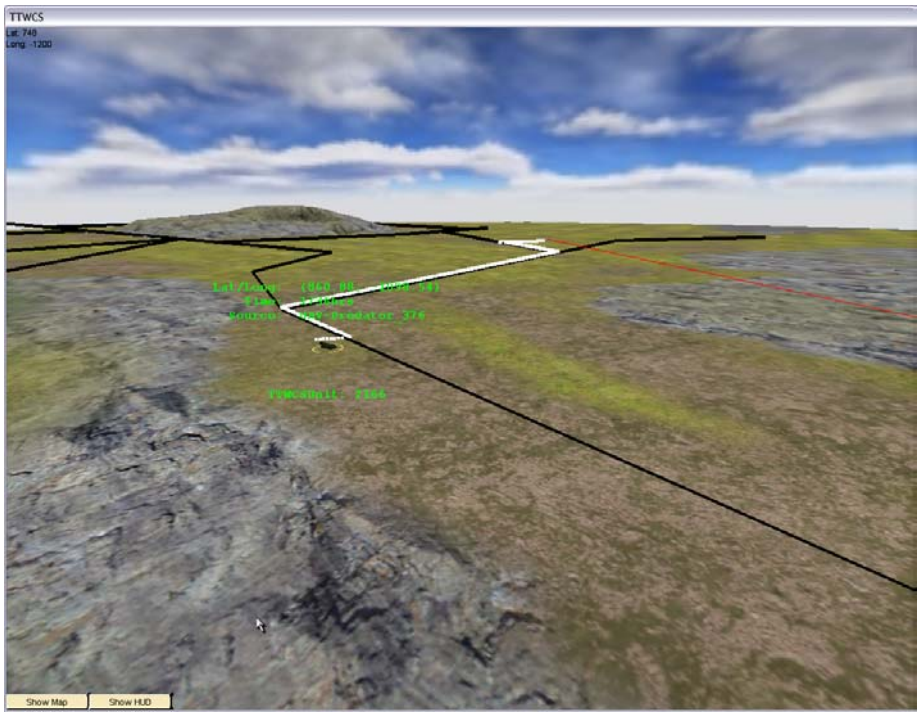


Figure 4 – Development of an NPC in AI.DE

One of the strengths of this approach is that the cost function can be comprised of a number of factors. For example, adding a cost factor for *exposure to danger* would be useful in the TTWCS application. We expect, for example, that the hostile ground vehicles will often choose their routes based upon well they can hide from satellite and airborne sensors. In Phase II we plan to include these factors in our pathfinding improvements.



The next step was to add in a road network to see how well A* could quickly and accurately compute the best route across a combination of ter-

Figure 5 – Terrain & Road Network Path-Following in Torque Engine

rain and roads. As part of this effort we decided to switch from the basic "first person shooter" (FPS) version of *Torque* to the "real time strategy" (RTS) version to better handle larger map areas. After overcoming some minor integration issues, we successfully added in the roads and tested A*'s ability to use them as part of its route building.

Figure 5 is a screenshot taken during this activity. In this example, the path illustrated is that from a TEL, located on a grassy area off the road, to a predicted destination which is also located some distance off-road. The white highlighted route indicates the most likely path, with the vehicle first traveling to the nearest road and then following the road network to a point close to the destination, with a final leg across open terrain.

7. Vehicle Physics

Knowing where a vehicle might move next is not enough: we must also be able to predict how *quickly* it can move from one location to another. One way of doing this is by running what are termed "vehicle mobility models"; in our Phase I proposal we discussed our plans to investigate the use of a modern game engine or its standalone physics engine to achieve results equal or better than traditional mobility models. This viewpoint was based on the recent success of games like *Gran Turismo 4*, which have taken their physics modeling of vehicles to the point where they can accurately simulate the real-world performance of vehicles down to minute detail, such as tire pressures *vs*. road surfaces, etc.

In Phase I we started this aspect of the research with the *Havok* physics engine, as it appeared to provide the most comprehensive vehicle modeling tools. We soon ran into licensing issues and had to discard the *Havok* solution. We found the next best alternative to be the *Novodex* engine. This decision was partly influenced by the fact that Epic Games had recently chosen Novodex for its next version of the *UnReal* Engine, and bolstered by the fact that *Novodex* offered an excellent evaluation package at no cost.

Figure 6 – TEL Vehicle Simulation in ODE

Although *Novodex* didn't provide a separate vehicle package, as we began working with its development environment we found it relatively easy to assemble basic components into workable vehicle simulations; enough to show the feasibility of our initial concept. The expiration of our *Novodex* evaluation license, combined with the specter of continuing to face heavy license fees, drove us to our next platform: the *Open Dynamics Engine* (ODE), which is available as open-source. While we didn't have sufficient time in Phase I to advance the vehicle modeling with ODE any further than we had with Novodex. We did reach the conclusion that the development environment was adequate for our purposes and integrated well with the Delta3D game engine. This included three dimensional models of vehicles, as shown in Figure 6, which illustrates a SCUD launch vehicle, implemented in ODE, attempting to climb a steep hill. While our model thus far is using generic values for weight, center-of-mass, etc., we plan to increase the fidelity in our next phase.

8. Visualization

Although our approach of using gaming technology was based primarily on the industry's great strides in artificial intelligence and vehicle simulation, the compelling 3D visualization of the game engines promised to provide an additional "bonus" by adding to the situational awareness of the operator. While 2D views of the battlefield are sufficient for most decision-making, we believe the ability to also "fly" through the target area and see the layout of the terrain will give the TTWCS operator much better insight into where a target might move or how a threat might be positioned.

For this portion of our investigation we made the assumption that a typical area of interest would be approximately 100 miles on a side. Beginning with the *UnReal* engine we found that it has a maximum gam-

ing area of approximately 4 miles on a side, expressed as 524,288 (512K) units. While this gives excellent resolution (each unit equal to about a half-inch), the gaming area is too limited for our purposes. We attempted to change the scaling to increase the boundaries but ran into performance and stability problems. In looking at UnReal developer forums on the web, we found others have had similar experiences. We then moved on to the *Torque* engine and had similar difficulties.

Figure 7 – Large-Scale Terrain Simulation

We next tried working with *Flight Simulator* from Microsoft, as it has essentially an unlimited gaming area, achieved by using generic blocks of terrain for everything but airports and their immediate surroundings. Using the API that Microsoft offers, we were able to replace any set of generic terrain with more detailed data. We obtained this new terrain data from the U.S. Geological Survey (USGS) website [x] which has an excellent selection of unclassified elevation and feature/texture data. We prototyped our concept by selecting the area around AVI (Northport, NY) from the USGS website and replacing the corresponding area in *Flight Simulator*. Figure 7 is a screenshot taken during this activity; Northport Harbor is visible in the lower right, just below the plane's wingtip:

While this experiment in Phase I proved the feasibility of simulating large, custom terrain via a game engine, the MS Flight Simulator package has other shortcomings in terms of AI and physics.

For Phase II we plan to resolve the large terrain problem by utilizing the *Delta3D* game engine, being developed at the MOVES Institute of the Naval Post Graduate School. Delta3D has the ability to process and display extremely large terrain areas, and is based on an open-source architecture that promises to be an excellent fit to all of our requirements.

9. Training

The use of gaming technology for the TTWCS application brings with it the added benefit of providing a natural foundation for developing an embedded training mode. One of our Phase II objectives is to add operator training features that can be utilized either in a stand-alone mode or as a participant in networked training exercises.

Working with the MOVES Institute we will design and implement a set of training features to be embedded into the new system software. Most of the requisite elements, such as simulation of the opposing forces, will already be in place in the form of the artificial intelligence and physics engines. In addition, we plan to add several training-specific elements, including:

- Simulation of Tomahawk missile fly out
- Sample lesson content
- Pass/fail criteria
- Capture of trainee actions
- Grading or after-action review of trainee actions
- Tracking of trainee progress through lessons

The training tasks will include basic set up and use of the system, as well as a limited number of scenarios aimed at teaching the fundamental workflow and methodology of analyzing Tomahawk and target parameters and relationships. In addition, an HLA-compliant interface may be added to allow the ability to participate in networked training exercises.

While this capability will initially be built into the SBIR prototype, we foresee it evolving later into forms that will run on a variety of platforms including consoles such as X-Box. This evolution will allow the TTWCS operators to train/play in their off-duty hours, working with a larger number of scenarios, targets, and terrain, perhaps even competing with each other in mock attacks to sharpen their skills.

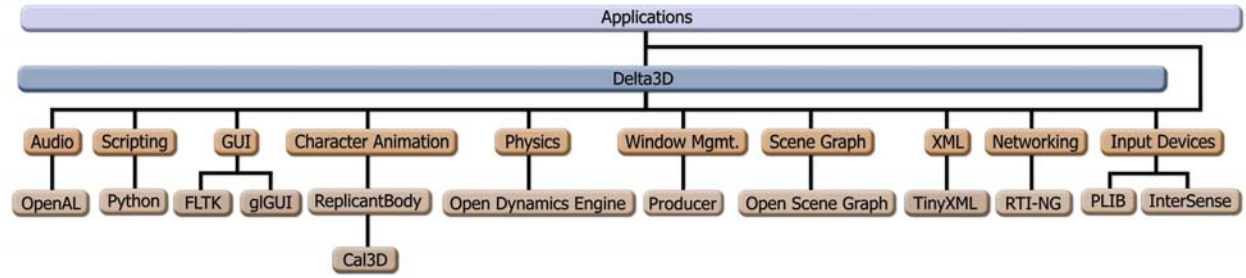


Figure 8 – Delta3D Architecture

10. Delta3D

Delta3D is an open source game engine designed specifically for military games and simulations. Historically, using a game engine has required significant financial resources. However, Delta3D is specifically designed to provide a low cost, easy to use alternative to commercial and proprietary game engines. While in the past, using an engine for certain applications might have been too cost prohibitive, now an engine can be used for these products. These include several different types of applications:

- those where graphical quality enhances value but is not the major consideration;
- smaller applications where each user might only use the application for a short period of time, such as fifteen minutes to one hour;
- those produced by developers lacking the capital to invest in a commercial engine.

While Delta3D does not require high licensing costs, it is a full featured engine easily capable of serving as the base for most applications. Additionally, it has several advantages over proprietary engines. These engines often require data to be in unique formats which cannot be used in other engines. This locks the developer into using that engine for all future modifications whether that engine is the best fit for them, effectively leaving the developer at the mercy of the engine builder. Additionally, if the proprietary game engine does not meet the developer's needs, they have no way to modify it. This leads to time wasting work-arounds for the developer, limiting productivity. McDowell *et al* [6] goes into more detail on the problems faced using proprietary engines and how they shaped the design philosophy of Delta3D.

Delta3D is made up of many other open source products which form the different modules of the engine. For example, Delta3D uses the OpenSceneGraph library for rendering, the Open Dynamics Engine for physics, Open AL for audio, etc. Delta3D acts as a thin API over these other systems, giving the developer a wide range of capabilities while only learning one API. Figure 8 (above) shows all the open source projects which currently make up Delta3D and what features each provides. Using these other projects as the basis of Delta3D allowed the engine to be built in a short period of time but nonetheless turn out extremely powerful. Additionally, this provides Delta3D with an inherited developer base, meaning that as these other source products improve, Delta3D also benefits from those improvements.

One feature of Delta3D which makes it particularly well suited for the TTWCS is its advanced method to render terrain. Delta3D can 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 [7], created by William Wells, an Air Force PhD student at MOVES. Wells' approach begins by processing elevation data points to create 1 degree by 1 degree skirted height field meshes of the terrain. The elevation data is imported directly from an elevation data repository (e.g. DTED™ – Digital Terrain Elevation Data from the National Geospatial-Intelligence Agency) at run-time. Detail between the known elevation postings are added by subdividing the base mesh and increasing or decreasing the values of the linearly interpolated midpoint heights with Perlin noise. The algorithm uses the SOARX continuous level-of-detail (CLOD) algorithm [8] to take the enhanced height field data and construct a dynamically optimized mesh grid based on the user's view frustum. As the user nears the edge of the existing terrain, we determine the next geocell's coordinates, load up the corresponding source data from our repository, and process the next geocell in the same manner. Generating a geocell's height is very quick, on the order of a few seconds. Because elevation data is available for most of the world, this algorithm can produce a near endless source of optimized elevation meshes from raw source data with increased resolution.

Satellite imagery is layered over this elevation data at run time with normal maps which represent the base and detail gradients which adds relief shading and surface details. Once a geocell's data has been calculated, it can be saved for improved load times. Of particular

interest to simulations is the ability to reproduce the exact same terrain by seeding the random generator with the same seed.

At this point, GENETICS adds realism to the terrain by vegetation using GeoTIFF (Geographic Tagged-Image File Format) files representing land cover data. In the U.S., the National Land Cover Dataset (NLCD) fills this requirement, while other systems exist in other parts of the world (i.e., CORINE). The data from the GeoTIFF Images is used to place the correct number and type of vegetation onto the terrain. Once the correct models representing the vegetation have been determined, they are added to the scene using an efficient data structure, such as a quadtree. This controls each vegetation object so it is drawn in a realistic and efficient manner.

This generation of both terrain and vegetation results in GENETICS and Delta3D providing applications with exceptionally realistic terrain. Such realistic terrain is one of the major features which make Delta3D ideal for a project such as the TWCS.

10. Summary

The Phase I SBIR described in this paper proved the feasibility of using gaming technology for predicting future movement of ground vehicles. Modern game engines incorporate sophisticated algorithms for artificial intelligence, path finding, physics and 3D rendering, all of which are directly applicable to this problem space. Future research in this area will expand the operational and training use of gaming technology for TTWCS, including use of the Delta3D engine being developed by the MOVES Institute.

11. References

[1]*United States Navy Fact File: Tomahawk Cruise Missile.* 11 Aug. 2003. Office of U.S. Navy Information. 4 Feb. 2004

<http://www.chinfo.navy.mil/navpalib/factfile/missiles/wep-toma.html >.

[2]Morrow, Capt. Steve. *What Comes After Tomahawk?* Naval Institute Proceedings, July, 2003.

[3] "*Terrain Trafficability in Modeling and Simulation*", Dr. Paul A. Birkel, MITRE Corp., SEDRIS Conference, Orlando FL, 2002.

[4] "*Optimizing Pathfinding",* Sean Barrett, Game Developer Magazine, a series of 5 articles running from Jan through May, 2005.

[5] On the web at http://seamless.usgs.gov

[6] *Military Uses of an Open Source Game Engine*, Perry L. McDowell, *et al*, Proceedings of the 2005 IMAGE Conference. Tempe, Arizona: The IMAGE Society.

[7] *Generating Enhanced Natural Environments and Terrain for Interactive Combat Simulations (GENETICS),* William D. Wells & Christopher J. Darken, Proceedings of the 2005 IMAGE Conference. Tempe, Arizona: The IMAGE Society.

[8] *Real Time Visualization of Detailed Terrain*, Andras Balugh, Master's Thesis, Budapest, Hungary: Budapest University of Technology and Economics.

12. Author Biographies

KEN DORIS is the Vice President of Engineering at Applied Visions, Inc. in Northport, NY. He serves as the Principal Investigator on the Navy SBIR project described in this paper as well an on-going Army SBIR that also uses gaming technology for battlefield analysis and visualization. One of the authors of IEEE 1278 (the original DIS specification), Ken has published numerous technical papers on subjects such as 3D visualization, real-time network traffic analysis and multicast addressing. He received his Bachelor of Electrical Engineering degree from Rensselaer Polytechnic Institute.

MARK LARKIN is a Project Engineer at Applied Visions, Inc. in Northport, N.Y. and is the Project Engineer on the SBIR project described in this paper. He has over eighteen years experience developing Visualization and Modeling system software. He was the lead developer on prior SBIR projects at AVI, including AF98-021 "Air Tasking Order Visualization – Applications of VR to AOC's and a DARPA funded SBIR project, SB992-043 "Visual Representation of Cyber Defense Situational Awareness". This project led directly to our current *SecureScope*^{TM} product line. Mr. Larkin holds both a Masters of Science in Computer Science from the New York Institute of Technology and a Bachelor of Electrical Engineering from the State University of New York at Stony Brook.

DAVID SILVIA David Silvia is an Engineer with the Naval Undersea Warfare Center (NUWC) in Newport, Rhode Island. He currently serves as a liaison for the Tactical Tomahawk Weapon Control System (TTWCS) Advanced Concept Working Group (ACWG) participating in technical exchanges between Naval Surface Warfare Center, Johns Hopkins Applied Physics Laboratory, Lockheed Martin Corporation, and NUWC. In addition, Mr. Silvia conducts technical evaluations, surveys new candidate technologies, and develops prototypes to address future requirements for the TTWCS. His past work for the Navy has included the development of applications in speech recognition, distributed database design, and peer-to-peer architec-

tures. Mr. Silvia has over 20 years of experience in software development and engineering. He has Bachelors Degree in Engineering from Roger Williams University in Bristol, Rhode Island. He has worked for the Navy for five years and lives in Massachusetts.

PERRY MCDOWELL is a former Naval Nuclear Power Surface Warfare Officer. He served as Damage Control Assistant on USS VIRGINIA (CGN-38), Operations Officer on USS ELROD (FFG-55), and Reactor Controls Assistant and Main Propulsion Assistant on USS ENTERPRISE (CVN-65). He has been on the faculty of the Naval Postgraduate School since 2000, where he teaches computer science and does research in virtual environments and training for the Modeling, Virtual Environments, and Simulations (MOVES) Institute. His research has been primarily focused on training in virtual environments, and he is the Executive Director for the Delta3D open source game engine. He is currently conducting research for his PhD. He graduated with a B.S. in Naval Architecture from the U.S. Naval Academy in 1988, and earned an M.S. in Computer Science (with honors) from the Naval Postgraduate School in 1995. Mr. McDowell is pursuing a doctorate degree in Computer Science.