Building Training Games With the Delta3D Simulation Core

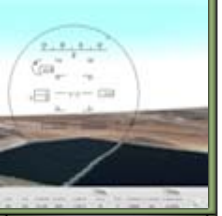
Alion Science and Technology
Naval Postgraduate School

Curtiss Murphy
cmmurphy@alionscience.com
Chris Rodgers
crodgers@alionscience.com
David Guthrie
dguthrie@alionscience.com
Perry McDowell
mcdowell@nps.edu

See Delta3D on the floor!

I/ITSEC Tutorial (Dec 1, 2008)

Booth 1215 - Alion
Booth 2923 - NPS





Tutorial Contents

- Intro
  - The Problem & The Solution
  - Game Engine Intro
- Tutorial Parts
  - Part 1 – Overview of Delta3D
  - Part 2 – The Simulation Core
  - Part 3 – Stealth Viewer
  - Part 4 – Driver Demo
  - Part 5 – Conclusion

TUTORIAL
PART 0

Introduction

Intro - Background

- Assumptions
  - Gaming technologies are a valuable part of our training toolbox
  - Delta3D interests you because it is Open Source
- 4th Annual I/ITSEC tutorial
  - See previous tutorials or references (end) for more info
- Constraints
  - Time limit – 90 minutes - topics covered briefly
  - Just try to get the ideas – slides are available
- Audience
  - Software developer or manager
  - Technical background
- Goal
  - Introduce how to create a full training app or game using the Delta3D Simulation Core

Intro – The Problem To Solve

- The obvious stuff…
  - Your customer needs a training game or modern 3D visualization tool
  - Your project is willing to invest in development
- But where do you start?
  - Dozens of engines (proprietary and open source)
    - Huge variety of out-of-the-box tools
  - But what if they don’t meet your needs?
  - I don’t want to start from scratch!
- The solution …






Intro – The Solution

- Delta3D Simulation Core
  - Base library provides common M&S functionality
  - Ready to go – don't start from scratch
  - Open source
  - Widely used
  - Working, out of the box HLA applications
- Let's back up a minute…






Intro - What is a Serious Game?

- Serious Game
  - Use of game technology for non-entertainment purposes (ex. Training)
- Why use Serious Games at all?
  - Experiential fidelity!
    - Dynamic Interaction
    - Engaging Immersion
    - Simple Interface
    - Interesting Decisions






Intro – What is a Game Engine?

- Visualization
  - Move around and ‘see’ the simulation – typically 3D
  - 3D Models (trucks, planes, tanks, soldiers)
  - 2D Textures (brick walls, satellite imagery, UI Icons)
  - Terrain (large or small, indoor or outdoor)
  - Shaders (detail mapping, specular highlights, bump maps)
- Behaviors
  - Moving and rotating in 3D space
  - Character Animation (walking, running)
  - Physics (collisions, gravity)
  - Weather (clouds, fog, sun rise and set)
  - Particle Effects (smoke, explosions)
- Misc
  - User Interface (Heads Up Display)
  - Input (joystick, keyboard, mouse)
  - Sound (voices, explosions, music, ambient)
  - Tools (editors, export/import, level design)





Intro – Tutorial at a Glance

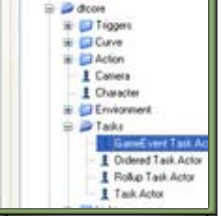
- Part 1 – Overview of Delta3D
- Part 2 – Simulation Core
  - The Building Blocks for your Game
- Part 3 – Stealth Viewer
- Part 4 – Driver Demo
  - Example multiplayer vehicle simulation
- Part 5 – Conclusion

- But first… a demo

Demo Time!

TUTORIAL
PART 1

Overview of Delta3D






Part 1 - What is Delta3D?

- Delta3D
  - Open Source Gaming Engine == FREE
  - Government maintained - Naval Postgraduate School (NPS)
  - Active community involvement
    - www.delta3d.org
    - ~ 2033 registered users, 14385+ Forum Posts, 55+ Tutorials
    - Dozens of companies and organizations involved
- What's it for?
  - Primarily for 3D visualization (such as Stealth Views)
    - 3D Models, 2D Textures, Input Devices, Audio, Physics, Weather, Terrain, Character Animation, Particle Effects, Graphics Shaders, User Interface (HUD)
  - Game-based training – especially Modeling & Simulation
    - HLA, After Action Review (AAR), Large Terrains (Terra Page, OpenFlight), Learning Management System (LMS), 3D Simulations
- Specifically geared to M&S community!





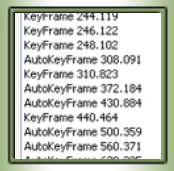

Part 1 - Legal Mumbo Jumbo

- Delta3D & SimCore licensed under LGPL
  - Lesser GNU Public License
    - http://www.gnu.org/copyleft/lesser.html
  - Non-viral in nature. Applications built with Delta3D may retain a proprietary license.
- STAGE & Stealth licensed under GPL
  - GNU Public License
    - http://www.gnu.org/copyleft/gpl.html
  - Modifications directly to STAGE's UI code may *not* retain a proprietary license unless you purchase a QT license.
  - Mainly because these are executables




Part 1 - Delta3D Features

- Cross platform
  - Windows XP/Vista, Linux, and Apple Mac OS X (unofficial)
- High level C++ API
  - Includes some Python bindings
- GameManager
  - Basic game management – actors, messages, & components
- After Action Review System
  - Record/playback and task tracking
- Networking Support
  - HLA, DIS, & game-style client/server
  - DDM subscription
- 3D audio
- Graphics
  - Full support for OpenGL Shading Language
  - Supports many 3D file formats (.3ds, .flt, .osg, .ive, .obj, Terrex)
  - 3D content exporters for Max 9, Maya 6, and Blender 2.x



Part 1 - More Features

- Learning Management System (LMS)
  - SCORM 2004 compliant
- Dynamic Actor Layer
  - For creating and setting properties on Actors
- Articial Intelligence (AI)
  - Pathfinding, planning, state machine, and sensors
- NVidia® PhysX™ Integration
  - PhysX™ distributed as part of Delta3D extras
- Character animation
  - GPU/CPU skinning, 3DS Cal3D plugin, animation blending
- Extensible terrain architecture
  - Support for DTED, SOARX LoD, procedural vegetation, GeoTIFF satelite imagery
- Tool suite
  - Level Editor (STAGE), Character Animation Editor, Particle System Editor, 3D Model Viewer, Waypoint explorer
- Unit tests (45,000+ lines)






Part 1 - Limitations of Delta3D

- How do you get started?
- The number of libraries can be confusing
- Limited pool of art assets (out of box)
  - Organizations don't always share
- Complex build procedure
  - Availability of source plus reuse of other open source libraries makes it more complicated to get it all setup to compile
  - Docs & tutorials available
- Limited lighting model & shadows
  - Can be solved per project

TUTORIAL
PART 2

Simulation Core

Part 2 – Simulation Core

- What is Simulation Core?
  - A large library of basic behaviors typically needed by most Simulators
  - Sits on top of Delta3D
    - Uses all major Delta3D features
  - Part of the Delta3D-extras repository

https://delta3d-extras.svn.sourceforge.net/svnroot/delta3d-extras/SimulationCore/trunk
https://delta3d-extras.svn.sourceforge.net/svnroot/delta3d-extras/dtAgeiaPhysX/trunk

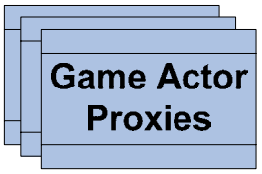
- Provides most features you need to build a networked simulation application
  - Customizable
    - Open Source - Can modify Simulation Core code
    - Or use Actor libraries to add custom vehicles, UI, and other behaviors without modifying Sim Core or Delta3D
- Includes a fully functional Stealth Viewer
  - Typically used as the instructor operating station



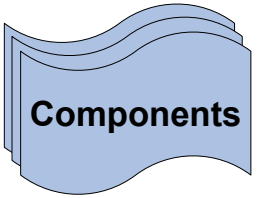



Part 2 – A Quick Review (1)

- Quick reminder of Delta3D architecture
  - See Delta3D tutorials or prior I/ITSEC materials
- Actors
  - Objects in the world that we care about
  - Can be moving (a tank) or static (a tree), visible (missile) or invisible (daytime)
  - Almost anything can be an actor
  - Has properties and is loaded as part of a map (xml)



- Components
  - High level behaviors
  - System level stuff - networking, logging, HUD, input
  - Receives all messages from Game Manager

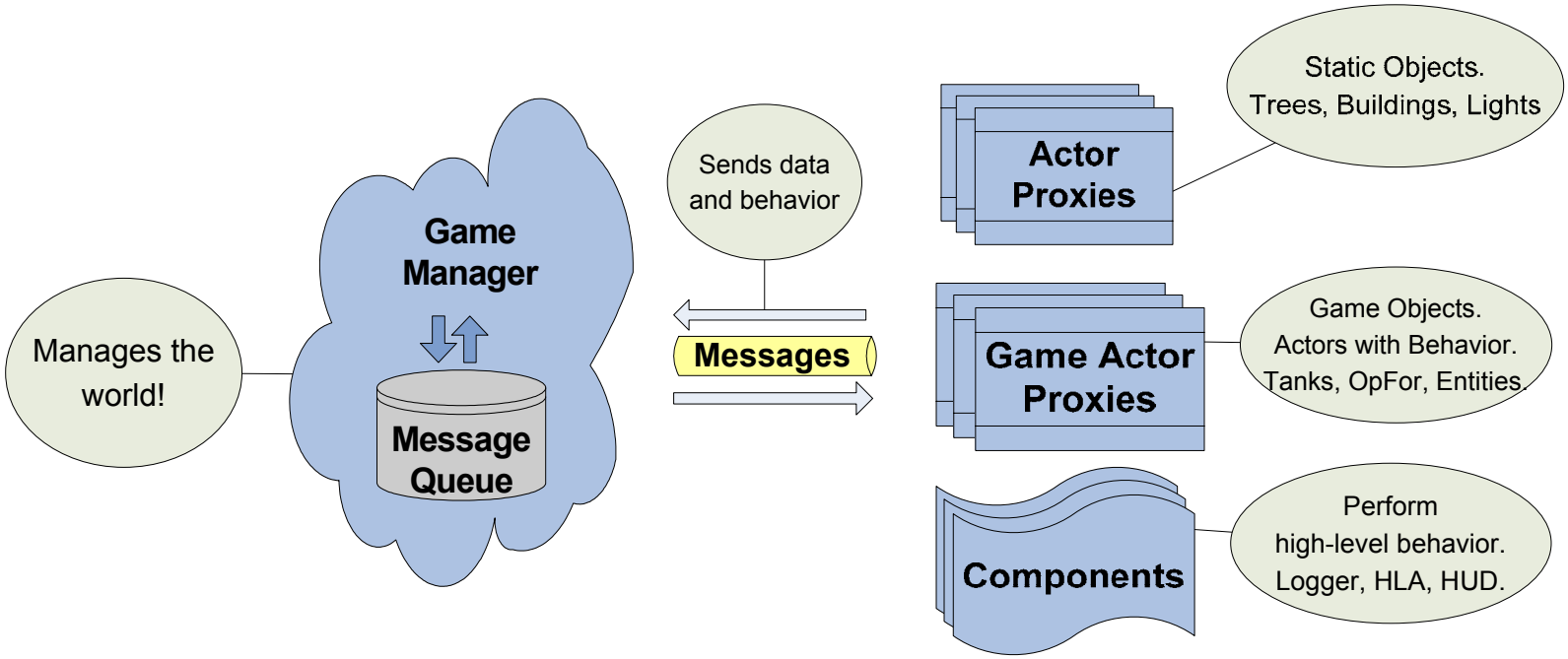






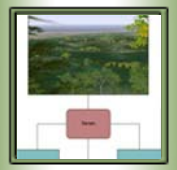
Part 2 – A Quick Review (2)

- Game Manager
  - Manages all Actors and Components
  - Primary job is to route messages
  - Controls time and ticks (via messages)
- What it looks like:



Part 2 – Networking

- Why was that quick review important?
  - Sim Core is almost entirely Actors and Components
  - Your game will be too!
- HLA Networking
  - RPR 1 & RPR 2 ready – just add an RTI
  - Tested with RTI-S & RTI-NG Pro & CERTI
  - Configurable settings - understands connect/disconnect
  - Drives almost all features in Simulation Core
- Dead Reckoning & Ground Clamping
  - Remote entities smoothly dead reckons position – handles large changes
    - Supports linear velocity/acceleration & angular velocity/acceleration
  - Variable level of detail (more up close, less far away)
  - Ground clamping - multi-point, single point, intermittent, or none (flying)
  - Local vehicles configured to compare current position vs DR position before publishing entity updates



Part 2 – Visualization

- Terrain
  - System designed to expect a ‘terrain’
  - Supports multiple terrain formats (Terrapage, Openflight, IVE – others can be customized per project)
- Particle System Management
  - Long duration effects for smoke & munitions
  - Affected by current weather conditions (wind affects particles)
- Weather Visualization
  - Clouds, time of day/year, rain, snow, nighttime visibility
  - Sunrise & sunset
- Dynamic Lights
  - Lighting for flares, explosions, burning vehicles, headlights, tracers, etc…
  - Closest to camera (N possible limited by your hardware)
- Audio effects
  - Vehicle idle sounds, munition fire, impact, & detonation sounds
- Rendering Support Component
  - Supports post process effects





Part 2 – Munitions

- Supports both Remote (interactions) and Local (weapons)
- Physics based weapon firing system
  - Each bullet individually modeled by physics munition particle system
- Direct & Indirect fire, detonations, particle effects, smoke trails
- Tracer, flash, and lighting effects
- Applies physics (force) to local vehicles
- Customizable – munition types in XML
  - Types, damage models, particle systems (including physics), force, lighting





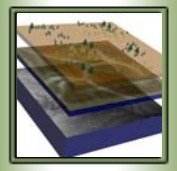

Part 2 - Entities

- Integrated physics engine (PhysX™)
  - Base physics vehicle & physical munitions
- Articulations
  - Supports both Remote and Local articulations on models
  - Ex. Gun swiveling on turret
- Entity Types
  - Four wheeled vehicle, base physics vehicle, flare, missile (with smoke trail), generic platforms (fly or ground)
  - Human character - hardware skinning, animation blending, & animation transitions
- Add your own via custom Actor Libraries …

Part 2 – Content Assets

- Open GL Shaders Examples
  - Vertex and fragment shader behaviors
  - Per pixel dynamic lighting, Ephemeris weather effects, emissive particles, fish eye lens, specular hightlights and reflective surfaces, tracers (volumetric lines)
- Base Audio Examples
  - Ground and vehicle impacts, multiple explosions, car brakes, multiple weapons fire, some vehicle sounds
- 3D Model Library
  - A limited variety of 3D models available in SimCore and Delta3D art asset repositories



Part 2 – Misc Features

- Heads Up Display Elements
  - Compass control, data field, GPS controls, speedometer,
- Coordinate Conversion
  - Lat/Lon, MGRS, and XYZ Cartesian
- Camera Motion Models
  - Clamped for vehicle, overhead flight for Stealth Viewer, FPS for human walking
- XML configuration - maps & data files
- User Tools
  - Binoculars, Compass, GPS, and Help





Part 2 – GOTS Features

- These are features which already exist, but may require special access or permissions
  - Provided as information only.
  - Contact Curtiss Murphy for more info.

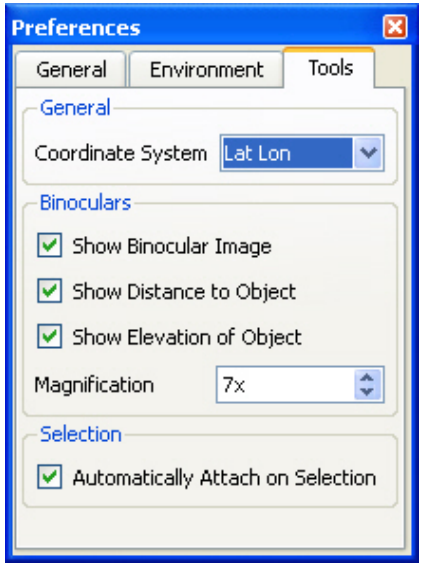
- Integrated with Common Sensor Model (CSM)
  - Realistic night vision and other sensors behavior
  - Sensor washout & high dynamic rendering
- Road-Based Dead-Reckoning
  - Reduces network traffic by DR'ing along roads
- Large collection of assets
  - US 3D models, IR models, sounds, particles, …
- 2D map tool

TUTORIAL
PART 3

Stealth Viewer

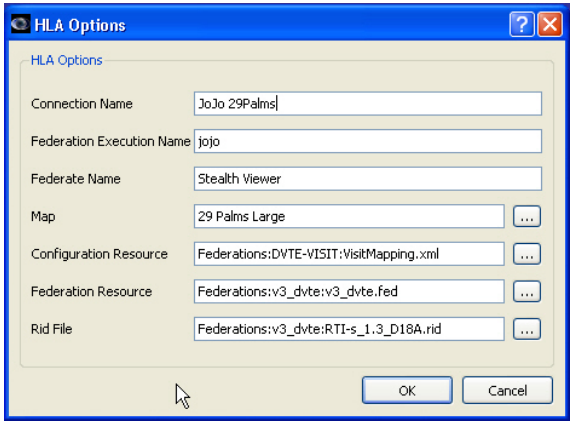
Part 3 – Stealth Viewer

- Part of Simulation Core
  - View any sim core app!
- Commodity Stealth features
  - 3D View
  - Entity Search Tools
    - Attach (follow) or get entity details
  - Binoculars, Help, NVG, Compass
- Full Featured User Interface
  - Remembers user preferences
  - HLA connection interface for multiple networks & easy connection



Part 3 – After Action Review

- Record and Playback
  - Built in, stand-alone, easy to use AAR
  - Record the simulation as it happens
  - Replay behaviors from any angle/location
  - Playback without a network
- Instructor World View
  - Anywhere, any time, any angle



Demo of StealthView

TUTORIAL
PART 4

The Driver Demo

Hint: Code Snipets…






Part 4 – The Driver Demo (1)

- Ready to use example Vehicle Simulator
  - Completely functional, physics based vehicle simulator
    - Open Source – all source available to guide developers
  - Multiplayer - networked (HLA) with remote and local entities
  - Can take damage and fire weapon systems
  - Publishes self over HLA – correlated with dead reckoning
  - Demonstrates proper use of Simulation Core
- Supports two vehicle types
  - Four Wheel Vehicle from Sim Core
  - Hover Vehicle – example custom vehicle
    - Physics based floating vehicle
    - Applies forces to move – simulates wind resistance
    - Responds to explosions
    - Articulates turret (locally and remote)
    - Extends BasePhysicsVehicleActor






Part 4 – The Driver Demo (2)

- Example behaviors
  - Custom Actor Library
    - Registered in DriverActorRegistry
  - Game Entry Point
    - Load maps, configure components, command line params, start vehicle.
  - Heads Up Display Component (UI)
    - Uses several controls - Crazy Eddie GUI (CEGUI)
  - Input Component
    - Traps key inputs
    - Manages vehicle & weapons
    - Allows player to fire weapon
  - Game App Component
    - Start up conditions – simplifies GameEntryPoint

Part 4 – HUD Component Snippets

- Example – creating compass Control

```
// Compass Meter
mCompassMeter = new SimCore::Components::
        StealthCompassMeter("DriverCompassMeter");
mCompassMeter->Initialize();
mCompassMeter->SetPosition(leftOffset/SCREEN_WIDTH, 0.0f );
mCompassMeter->SetAlignment(
        SimCore::Components::HUDAlignment::LEFT_BOTTOM );
hudOverlay.Add(mCompassMeter.get());
```

- Example - setting HUD values

```
// Set the time control to the basic sim time
mSimTimeMeter->SetText2(dtUtil::DateTime::ToString(
      GetGameManager()->GetSimulationClockTime() / 1000000,
      dtUtil::DateTime::TimeFormat::CLOCK_TIME_24_HOUR_FORMAT) );

// Set speedometer
mSpeedometer->SetText2(dtUtil::ToString((int) vehicle->GetMPH()));

// Ammo meter
mAmmoMeter->SetValue(
      mWeapon->GetAmmoCount(), mWeapon->GetAmmoMax(), 0.0f );
```





Part 4 – Hover Vehicle Snippets (1)

- Example – Entering the World

```
void HoverVehicleActor::OnEnteredWorld()
 {
   // Create our physics shape – the helper does this
   GetHoverPhysicsHelper()->CreateVehicle();

   // We now have a physics object we can manipulate!
   NxActor *physActor = GetPhysicsHelper()->GetPhysXObject();

   // We want collisions and post physics updates
   if(!IsRemote())
   {
     GetHoverPhysicsHelper()->SetAgeiaFlags(
       dtAgeiaPhysX::AGEIA_FLAGS_GET_COLLISION_REPORT |
       dtAgeiaPhysX::AGEIA_FLAGS_POST_UPDATE);
   }

   SimCore::Actors::BasePhysicsVehicleActor::OnEnteredWorld();

   // Make remote entities non-kinematic so we bounce off remote vehicles
   if(IsRemote() && physActor != NULL)
     physActor->clearBodyFlag(NX_BF_KINEMATIC);
 }
```

Part 4 – Hover Vehicle Snippets (2)

- Example – handling vehicle controls

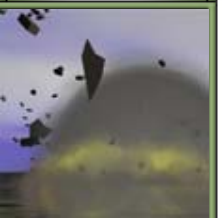
```
void HoverVehicleActor::UpdateVehicleTorquesAndAngles(float deltaTime)
 {
      …
      // FORWARD OR BACKWARD
      if (keyboard->GetKeyState('w') || (keyboard->GetKeyState('W')) ||
          keyboard->GetKeyState(osgGA::GUIEventAdapter::KEY_Up))
        accelForward = true;
      else if (keyboard->GetKeyState('s') || keyboard->GetKeyState('S') ||
          keyboard->GetKeyState(osgGA::GUIEventAdapter::KEY_Down))
        accelReverse = true;
      …
      GetHoverPhysicsHelper()->UpdateVehicle(deltaTime, currentMPH,
        accelForward, accelReverse, accelLeft, accelRight);
```

- Example – configuring publish rate

```
SetTimesASecondYouCanSendOutAnUpdate(5.0f);
```







Part 4 – Physics Helper Snippets

- Sometimes, we use Helpers to bridge an Actor and a Component
  - Encourages Aggregation over Inheritance

- Example - Create our Vehicle – a Sphere

```
bool HoverVehiclePhysicsHelper::CreateVehicle()
{
   osg::Vec3 startVec = GetVehicleStartingPosition();
   NxVec3 startPos(startVec[0], startVec[1],startVec[2]);

   // The important line!!! Create a collision sphere!
   SetCollisionSphere(startPos, GetSphereRadius(), 0,
      mVehicleBaseWeight, 0, "Default", "Default", false);

   // Reorient physics to our Y is forward system
   …
   return true;
}
```

Part 4 – Physics Helper Snippets

- Example – apply a forward force

```
void HoverVehiclePhysicsHelper::UpdateVehicle(float deltaTime,
        bool accelForward, bool accelReverse, bool accelLeft, bool accelRight)
{
  …
  float speedMod = GetVehicleMaxForwardMPH() * GetForceBoostFactor();

  // FORWARD
  if(accelForward)
  {
    NxVec3 dir(lookDir[0], lookDir[1], lookDir[2]);
    physicsObject->addForce(dir * (weight * speedMod * deltaTime),
          NX_SMOOTH_IMPULSE);
  }
  …
```

- Example – jump up

```
void HoverVehiclePhysicsHelper::DoJump(float deltaTime)
{
     GetPhysXObject()->addForce(NxVec3(0.0, 0.0, 1.0) * 9.8 *
        GetVehicleBaseWeight(), NX_SMOOTH_IMPULSE);
}
```

CONCLUSION






Organizations Using SimCore

- A partial list of some of the known users…

- USMC - PMTRASYS & TECOM
  - Deployable Virtual Training Environment (DVTE)
- Navy - NAVAIR – Manned Flight Sim
  - MH-60R Tactical Operational Flight Trainer (TOFT) #3
- Navy – Naval Service Training Command
  - Damage Control & Flooding Trainer
- Army - PEOSTRI
  - Common Sensor Model including NVG
- Army - West Point – Stealth Viewer
- Joint Forces Command – Stealth Viewer
- JIEDDO – Counter IED Trainer – Tactical Gaming
- USAF – Air Refueling Trainer

- Plus a variety of companies…






References

- www.delta3d.org – Best Reference Material!
  - Tutorials, Knowledge Base, Forums
- Game Programming Gems 7
  - "**Support Your Local Artist – Adding Shaders To Your Engine**" Murphy. Mar 2008
- I/ITSEC 2007 Tutorial
  - "**Creating Low-Cost Game-Based Trainers with Delta3D**" Murphy, McDowell.
- I/ITSEC 2007 Paper
  - "**Are You Ready? The Open Technology Development Challenge"** (Honorable Mention)" Joshi, Murphy.
- I/ITSEC 2006 Tutorial
  - "**Building Game-Based Trainers with the Delta3D Game Manager**". Murphy, Johnson.
- Game Programming Gems 6
  - "**Exposing Actor Properties Using Non-Intrusive Proxies**". Campbell, Murphy. Mar 2006.

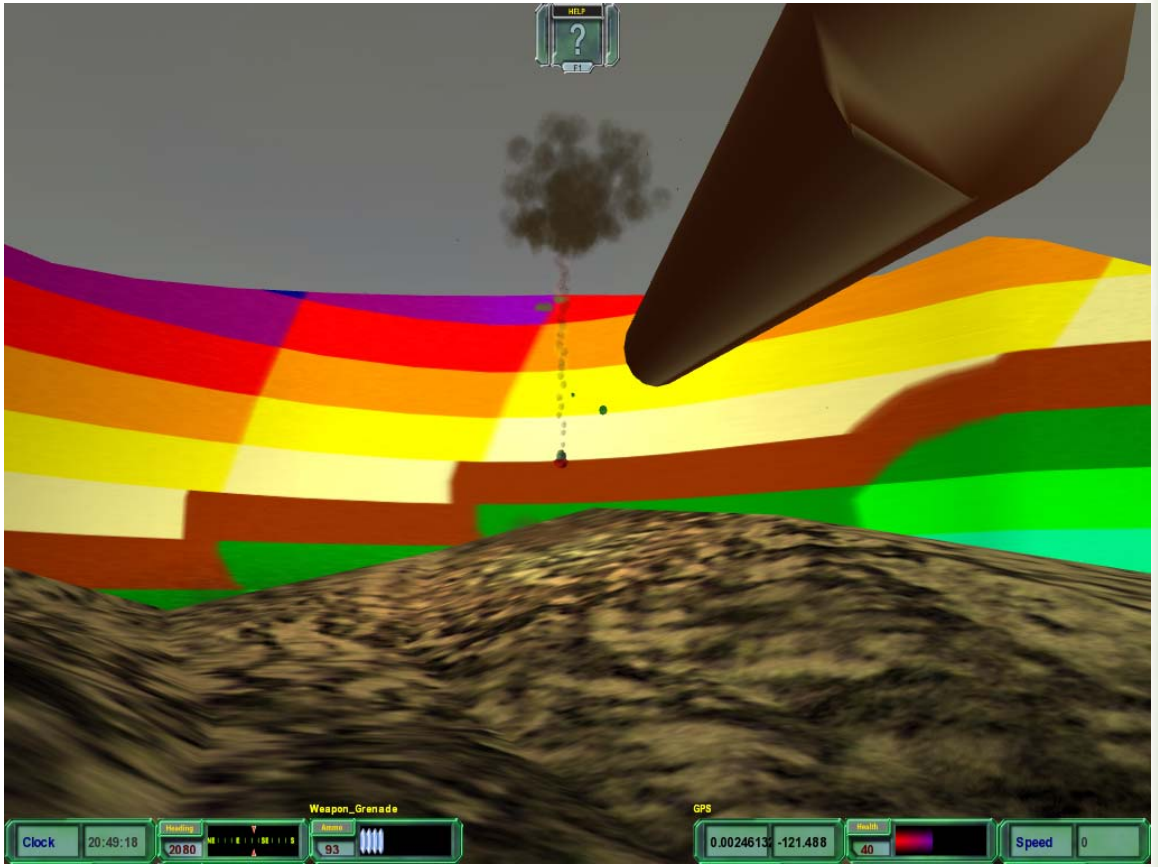
Final Demo and Questions

THE END

Building Training Games with the Delta3D Simulation Core

Thank you for attending our tutorial!

Please come to our booths to learn more about Delta3D!



STOP!!!

The Tutorial is over.

Go download the code!