Delta3D Stealth Viewer and DVTE build process

Version 0.2 - 5/9/2008

I. Introduction

This document covers downloading, building, and running the Delta3D Stealth Viewer and DVTE suite of applications. Four modules are part of the process: Delta3D, dtPhysX, SimulationCore, and DVTE-SimViewer.

- Delta3D is an open source simulation gaming engine available from SourceForge.
- dtPhysX is an integration of PhysX, the commercial physics engine from Nvidia, with Delta3D. Using dtPhysX requires installing OS Drivers and an SDK. It is not required for the Stealth Viewer, but the library SimulationCore on which the Stealth Viewer depends must be built with PhysX support in order the build the HMMWV driving sim from the DVTE suite. PhysX was formerly owned by Ageia until they were acquired by Nvidia, so the name Ageia will still be found in the library. dtPhysX refers to the sub-project itself, dtAgeiaPhysX refers to the repository directory name. dtPhysX is part of delta3d-extras on SourceForge.net.
- SimulationCore contains both the SimulationCore libraries and the StealthViewer. Some art assets and data files that could be released to open source are also included with it; however, to use US military platforms, integrate fully with JSAF, and enable night vision goggles and road-based dead-reckoning, the DVTE-SimViewer project is required. SimulationCore is part of delta3d-extras on SourceForge.net.
- DVTE-SimViewer contains the HMMWV driving sim, night vision goggle support, a fully integrated set of art assets and configuration data, and road-based dead-reckoning support.

The entire suite is supported on both Linux and Windows XP with the exception of the night vision goggle support, which has not yet been ported to Linux.

II. Build Environment

All of the sub-projects that make up the Stealth Viewer and DVTE Suite are built using CMake. This process typically supports both version 2.4.8 and 2.6, although we still recommend 2.4.8 if you are running on Windows due to some issue with debug builds. CMake is a powerful tool that allows you to define a single build process that can support builds on a wide variety of build environments. For this document, we will focus on Makefiles for Linux and Visual Studio 2005 (ver 8) for Windows. Visual Studio 2003 and 2008 should work, but all external dependencies will need to be rebuilt with the appropriate compiler. CMake is available at (http://www.cmake.org) – Downloads.

Note that although CMake is quite effective at generating cross platform builds, it may seem odd at times. For instance, sometimes, you will have to hit ‘Configure’ a number of times in a row.

III. External Dependencies

The first process for building delta3d is getting the external dependencies.

a. Windows

Go to www.delta3d.org, then downloads, then Delta3D external dependencies. You want the 2.0 package for Visual Studio 2005 (8.0).

You will still have to download and build Qt 4.3.4 (ftp.trolltech.com/qt/source/qt-win-opensource-src-4.3.4.zip). Open a command window and "cd" to the directory where you extract the zip file. Run configure.exe and then run nmake and nmake install, which comes with visual studio. You may need to run vsvars.bat or vcvars.bat from visual studio to setup your paths prior to running configure.exe.

b. Linux

Depending on the distribution used, different methods exist for installing the dependencies. A full source package is available on SourceForge.net for Delta3d 2.0. It has a combined build and install script that will generate an ext directory like the one for Win32. For many distributions, such as Fedora 8 and Ubuntu, most of the dependencies are available as prebuilt packages. The rest can be built from the source package. An easy way to do this is to edit the main script and comment out the scripts for the packages that are available on your distribution, then run it normally. The source package is also available from delta3d.org->Downloads->Delta3D External Dependencies. It is called dt_dep_src_2.0.tar.

Some Linux distributions have the environment variable QTDIR set by default. Make sure it is set to point to the 4.3.x version of QT. The value is /usr/lib/qt4 on Fedora 6 through 8.

IV. Checking Out the Source Repositories

The Delta3D 2.0 release has been modified on the trunk with changes that the Stealth Viewer and DVTE Suite require. Release branches also exist where the stable versions of the various sub-projects are matched up.

You will need a Subversion client to download the source. Tortoise works well on Windows. Typically the command line is used on Linux, but two plugins, Subclipse and Subversive, exist for the Eclipse IDE as well (http://www.eclipse.org) if you are planning on doing development. Both are good quality.

Do checkouts of the four repositories into the same directory. Make sure the names of the resulting directories match the names of the respective repositories, that is delta3d, dtAgeiaPhysX, SimulationCore, and DVTE-SimViewer.

The DVTE-SimViewer repository does not allow public access because it contains some sensitive data. You will need to contact Curtiss Murphy at Alion Science and Technology, cmmurphy@alionscience.com, for more information.

Trunk URLs

```
https://delta3d.svn.sourceforge.net/svnroot/delta3d/trunk/delta3d
```

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

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

```
https://svn.bmh.com/svn/DVTE-SimViewer/trunk
```

Recent Release Branch

```
https://delta3d.svn.sourceforge.net/svnroot/delta3d/branches/Alion_08_05_01_DVTE08_
```

```
Bugfix_RC
```

```
https://delta3d-extras.svn.sourceforge.net/svnroot/delta3d-
extras/dtAgeiaPhysX/branches/Alion_08_05_01_DVTE08_Bugfix_RC
```

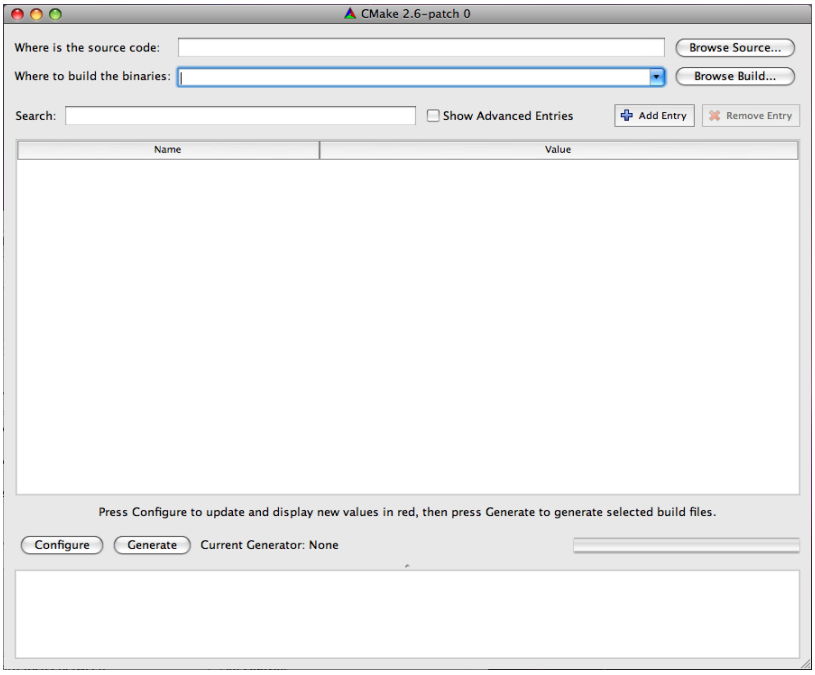
```
https://delta3d-extras.svn.sourceforge.net/svnroot/delta3d-
extras/SimulationCore/branches/Alion_08_05_01_DVTE08_Bugfix_RC
```

```
https://svn.bmh.com/svn/DVTE-SimViewer/branches/Alion_08_05_01_DVTE08_Bugfix_RC
```

V. Building Delta3D

Building Delta3D first requires running CMake. CMake may be run on the command line or as an interactive UI. A `configure` shell script exist to make this process simpler on Linux. If you edit that script to match your settings, you can run it once and be finished. In either the command line or UI case, CMake attempts to figure out the values for variables needed to generate the build system. The variables can be options, library paths, include paths, etc. These values may be set directly on the cmake command line using -Dname=value or in the UI. Here are the specifics for the UI with explanations for command line usage.

1. Run CMake (most likely, this is C:\Program Files\CMake 2.6\bin\cmake-gui.exe or CMakeSetup.exe).



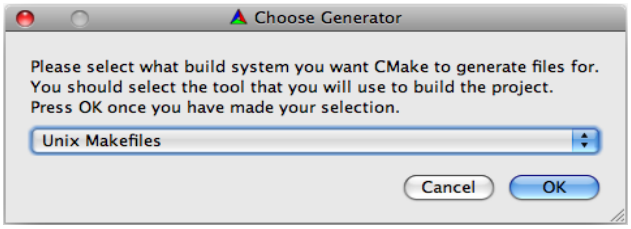
2. Set the source path and the build binary path. Both should be the path to the root of your Delta3D project (ie, something like C:\Projects\Delta3D). CMake can support having the source and binary paths being different, but it is easiest to keep them the same.
3. Once that is done, hit the configure button. It will then ask what generator to use.
    a) Windows

    On Windows, pick Visual Studio 2005.

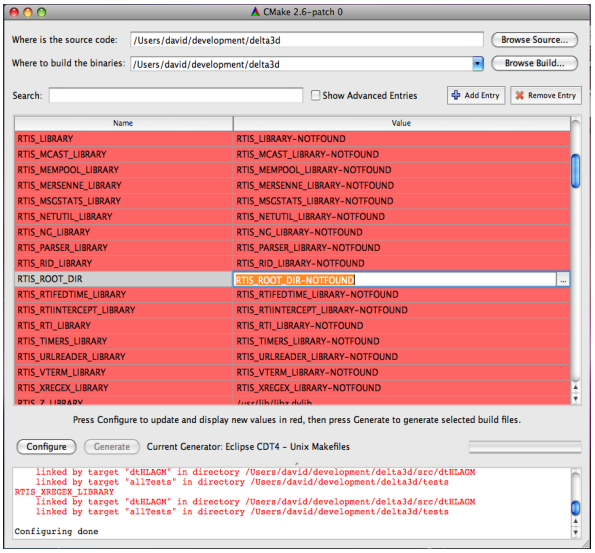
    b) Linux

    On Linux, pick Unix Makefiles. While other generators should work, they are not covered in this document. On the command line, the generator is selected using the -G option, e.g. `cmake . -G "Unix Makefiles"`. "Unix Makefiles" is the default on

Linux.



4. After clicking okay, CMake will run a first round of configuration. It will show what paths it found, what it didn't find, and what options are available. Everything will be highlighted in red at this point because the values are not considered final. As part of the configure step, it makes some effort to resolve some of the settings. It will look for a DELTA_ROOT and other settings. Look at each of the settings and paths and verify that they are pointing to the correct directories.
5. In addition to the overall settings, there are a few special settings that you need to consider. Look for the "BUILD_*" options in the UI. Note - if you are using the command line, you may set them with the "-D" option.
   5.1. BUILD_BINDINGS: Enable this, or set it to ON, if you want to build the python bindings. You don't need these for the StealthViewer or the DVTE suite, and they take a considerable amount of time to compile. We recommend you disable this.
   5.2. BUILD_EXAMPLES: Enable this, or set it to ON, if you want to build the delta3d example apps. These take time to build, and are unnecessary, but are helpful for development and also can be run to quickly verify if Delta3D is configured properly.
   5.3. BUILD_HLA: Enable this, or set it to ON, if you want to build the HLA components. This is required for the Stealth Viewer and DVTE Suite, but you will need to supply your own RTI. You will have to set a few paths in a future step.
   5.4. BUILD_ISENSE, BUILD_PLIB: These options deal with interfaces to joysticks and trackers. These are not used, so there is no need to enable them.
6. Run configure again.



7. Once configure finishes, you will see new variables for RTIS build options. If you have RTI and RTI_HOME set in your environment variables, it will probably find your RTI files. If these values are not correct, try setting the RTIS_ROOT_DIR option in the UI and press Configure

again. Note – if you are using the command line, simply add -DRTIS_ROOT_DIR=YourPathToRTI. Run configure again.

8. Hopefully, you now have no red values and no errors. Hit "Generate" or 'OK'.
9. Build Delta3D
    a) Windows

        On Windows, a "Delta3D.sln" file should exist in the root of the delta3d directory. Open your Visual Studio in the normal way and then open the solution file and build it.

    b) Linux

        On Linux, you should have a Makefile in the root. Running make (or make -j[x] where [x] is the number of concurrent threads), will build Delta3D. The default command line in the Linux `configure` script is:

        ```
        cmake -debug-output . -DBUILD_HLA:BOOL=ON -DBULID_PLIB:
        BOOL=ON -DBULID_BINDINGS:BOOL=ON -DBULID_ISENSE:BOOL=ON
        -DCMAKE_BUILD_TYPE=Release
        ```

        You should change it to

        ```
        cmake -debug-output . -DBUILD_HLA:BOOL=ON -DBULID_PLIB:
        BOOL=OFF -DBULID_BINDINGS:BOOL=OFF -DBULID_ISENSE:BOOL=OFF
        -DCMAKE_BUILD_TYPE=Release -DRTIS_ROOT_DIR=[Your rti-s
        path]
        ```

        If all the dependencies are setup correctly, this command line should run and build the Makefiles with no errors.

Note that the cmake build is only configured for use with RTI-s, with some support for CERTI being added. Supporting automatic configuration of other RTI's is something we would like to add. For now, the quickest way to add support for another RTI is to edit delta3d/CMakeModules/FindRTI-s.cmake. If you do add support for another RTI and would like to submit it, send the changed files and instructions to info@delta3d.org. The same is true for any Delta3d submissions.

VI. Building dtPhysX

This library is recommended but not required. You can build and run all the way up to the Stealth Viewer without using dtPhysX. SimulationCore and DVTE-SimViewer can build the parts needed for the Stealth Viewer without this library. However, if you plan on running the HMMWV application from DVTE-SimViewer, you must build this and compile support for it into SimulationCore. At that point, the Stealth Viewer will also depend on PhysX.

The process for dtPhysX is basically the same as delta3d above except that you point both CMake paths to your root dtPhysX directory. There are no build options to set, but you will need to install the PhysX System Software and SDK. Depending on which version or versions you have installed, CMake may not locate all of your directories. We recommend using PhysX version 2.7.3 at the moment.

For Linux, there is just one tar.gz file that extracts into a set of RPMS. Once you extract them, you can install them as root by typing `rpm -i *.rpm`, assuming you have no other rpms in the directory where you extract it. If you are on a system that doesn't use RPMS, go to `http://developer.nvidia.com/object/physx_downloads.html` and get the .deb version.

The CMake build will attempt to find the PhysX installation in the default locations. If CMake doesn't find it, you will need to set PHYSX_BASE_INCLUDE_DIR and possibly PHYSX_LIB_DIR_SEARCH. The base include directory is the parent directory for all the sub-library includes. It should contain the paths `Cooking/include`, `Foundation/include`, etc. Usually the directory is name `SDKs`. PHYSX_LIB_DIR_SEARCH should be found automatically on Windows since the libraries are under the `SDKs` folder. On Linux, it is `/usr/lib/PhysX` by the default. It should contain a version number directory such as "v2.8.0" which contains the libraries.

Also note that dtPhysX looks for delta3d in "../delta3d". If you don't have a DELTA_ROOT environment variable set and delta3d is not in the same parent directory as the dtAgeiaPhysX directory, you will have to set the DELTA_DIR CMake variable in the UI or on the command line with -DDELTA_DIR=[my path].

Assuming you have PhysX and Delta3D installed in the expected places, the `configure` script in the root should work for Linux without modification. You are now ready to build dtPhysX using Visual Studio (dtPhysX.sln) or make files.

VII. Building SimulationCore

Use CMake on SimulationCore the same way you did for Delta3D and dtPhysX. In addition to resolving your paths, there are 2 important build options:

1. BUILD_HLA: This option determines whether SimulationCore uses HLA or not. It defaults to ON or enabled. HLA is needed for the Stealth Viewer and some of the ther libraries, but you will have to provide your own RTI. See the section on building Delta3D for setting up the build with your own RTI. It works exactly the same way. If you do not enable this, the Stealth Viewer and HMMWV apps will not work. It is optional because some developers use various parts of the Simulation Core for non-HLA applications.
2. USE_PHYSX: This defaults to OFF. However, it is not required by the Stealth Viewer and when set to OFF, will save you from needing the PhysX installers when doing an installation. You must set this to ON in order to build the HMMWV application in DVTE-SimViewer. Note – you need to be consistent with this setting across all of your projects.

SimulationCore looks for Delta3D in "../delta3d". If you have not put it there, you will need to set the DELTA_DIR variable in the UI or with the "-DDELTA_DIR=[my path]" command line option. Likewise it looks for dtAgeiaPhysX in "../dtAgeiaPhysX" or "../dtPhysX". You can override that by setting DTPHYSX_DIR.

a) Windows

   On Windows, a “SimulationCore.sln” file should exist in the root of the SimulationCore directory. Open your Visual Studio in the normal way and then open the solution file and build it.

b) Linux

   Assuming you have Delta3D, PhysX, and dtPhysX installed in the expected places, the `configure` script in the root should work for Linux if you simply set `-DRTIS_ROOT_DIR=[Your rti-s path]`. That is, the full command line should like:

```
cmake -debug-output . -DBUILD_HLA:BOOL=ON -DUSE_AGEIA=ON
-DCMAKE_BUILD_TYPE=Release -DRTIS_ROOT_DIR=[Your rti-s
path]
```

or

```
cmake -debug-output . -DBUILD_HLA:BOOL=ON -DUSE_AGEIA=OFF
-DCMAKE_BUILD_TYPE=Release -DRTIS_ROOT_DIR=[Your rti-s
path]
```

VIII. Building DVTE-SimViewer

Use CMake on DVTE in the same way as Delta3D. It has two special build options.

1. BUILD_ROADDR: This defaults to OFF. Tum it on to build road-based dead-reckoning support into a plugin. On Windows, this is not working properly at the moment. On Linux, you must supply a path to JSAF. It will look in "../JSAF" and "../JSAF-2007". You can also just set JSAF_INCLUDE_DIR to the top root JSAF directory. In order to actually build the library, you have to generate a static library from a compiled JSAF. See `source/RoadDR/buildLibRoadDRFull.sh` where you have to set a path to JSAF and potentially edit the list of JSAF libraries to import. It will generate `lib/libRoadDRJsaf.a`.
2. USE_PHYSX: This defaults to OFF. However, it is not required by the Stealth Viewer and when set to OFF, will save you from needing the PhysX installers when doing an installation. You must set this to ON in order to build the HMMWV application in DVTE-SimViewer. Note – you need to be consistent with this setting across all of your projects. See Building dtAgeiaPhysX about setting up the paths to PhysX.

As usual, use the Configure option to make sure all of your settings line up appropriately. When you are ready, Generate and build.

a) Windows

   On Windows, a “DVTE.sln” file should exist in the root of the your DVTE-SimCore directory. Open your Visual Studio in the normal way and then open the solution file and build it.

b) Linux

   Assuming you have Delta3D, SimulationCore, PhysX, and dtPhysX, installed in the expected places, the `configure` script in the root should work for Linux with little modification.

   A full command line should look like this, depending on build options:

   cmake -debug-output . -DUSE_AGEIA=ON -DBUILD_ROADDR=ON
   -DCMAKE_BUILD_TYPE=Release `-DRTIS_ROOT_DIR=[Your rti-s path]`

We were able to achieve some significant performance benefits by using some code from Open Scene Graph 2.4 and by modifying the osgdb_txp plugin. Libraries for Windows are checked into DVTE-SimViewer\ext\lib\win32 and SimViewer\ext\bin\win32. They can be dropped into delta3d\ext\lib and delta3d\ext\bin respectively. These modified libraries are not required, but may improve performance on some systems, especially laptop computers. DVTE-SimViewer builds an osgdb_txp plugin, but we no longer use that code base. It will eventually be removed.

IX. Running Delta3D Applications

Once CMake is run on all the repositories, and built using CMake or Visual Studio, applications still will not run because all of the compiled libraries will need to be added to the library path.

a) Windows

The accepted solution on Windows has been to run Visual Studio from a batch file so that the path will be setup, but you also set the environment variables system-wide. Either way, you need to set DELTA_ROOT, DELTA_DATA, and PATH. Here is an example an DEV environment variable is set, and all the projects are installed there:

```
set PATH=%DEV%\dtAgeiaPhysX\bin;"C:\Program Files\Ageia Technologies\Ageia PhysX SDK\v2.7.3\Bin\Win32";%DEV%\SimulationCore\bin;%DEV%\DVTE-SimViewer\bin;%DEV%\delta3d\bin;%DEV%\delta3d\ext\bin;%DEV%\delta3d\ext\bin\osgPlugins;%DEV%\rtis-13\bin\win32_msvc-8.0;C:\Qt\4.3.4\bin

export DELTA_ROOT=%DEV%\delta3d

export DELTA_DATA=%DELTA_ROOT%\data
```

b) Linux

On Linux LD_LIBRARY_PATH needs to be set, as well as DELTA_ROOT and DELTA_DATA forthe Delta3D examples to work. Assuming the DEV environment variable is set, here is an example

```
export LD_LIBRARY_PATH=${DEV}/dtAgeiaPhysX/lib:/usr/lib/PhysX/v2.7.3:${DEV}/SimulationCore/lib:${DEV}/DVTE-SimViewer/lib:${DEV}/delta3d/lib:${DEV}/delta3d/ext/lib:${DEV}/delta3d/ext/lib/osgPlugins:${DEV}/rtis-13/lib/linux_g++-4.1

export DELTA_ROOT=${DEV}/delta3d

export DELTA_DATA=${DELTA_ROOT}/data
```