SmartBody Manual

Overview

SmartBody is a character animation platform originally developed at the University of Southern California.

SmartBody provides the following capabilities in real time:

- Locomotion (walk, jog, run, turn, strafe, jump, etc.)
- Steering - avoiding obstacles and moving objects
- Object manipulation - reach, grasp, touch, pick up objects
- Lip Synchronization and speech - characters can speak with lip-syncing using text-to-speech or prerecorded audio
- Gazing - robust gazing behavior that incorporates various parts of the body
- Nonverbal behavior - gesturing, head nodding and shaking, eye saccades

SmartBody is written in C++ and can be run as a standalone system, or incorporated into many game and simulation engines. We currently have interfaces for the following engines:

- Unity
- Ogre
- Panda3D
- GameBryo
- Unreal

It is straightforward to adopt SmartBody to other game engines as well using the API.

SmartBody is a Behavioral Markup Language (BML) realization engine that transforms BML behavior descriptions into realtime animations.

SmartBody runs on Windows, Linux, OSX as well as the iOS and Android devices.

The SmartBody website is located at:

http://smartbody.ict.usc.edu

The SmartBody email group is located here:

smartbody-developer@lists.sourceforge.net

You can subscribe to it here:

https://lists.sourceforge.net/lists/listinfo/smartbody-developer

If you have any other questions about SmartBody, please contact:

Ari Shapiro, Ph.D.

shapiro@ict.usc.edu

Downloading SmartBody

The SmartBody source code can be downloaded using subversion (SVN) on SourceForge. You will need to have a Subversion client installed on your computer.

Use the SVN command:

```
svn co https://smartbody.svn.sourceforge.net/svnroot/smartbody/trunk smartbody
```

The entire repository is fairly large, several gigabytes in size. Note that only the trunk/ needs to be downloaded. Other branches represent older projects that have been since been incorporated in the trunk.

Windows

We recommend using Tortoise (http://tortoisesvn.net/). Once installed, right click on the folder where you want SmartBody installed then choose 'SVN Checkout', then put `https://smartbody.svn.sourceforge.net/svnroot/smartbody/trunk` into the 'URL of Repository' field, then click 'Ok' - this will start the download process.

Updates to SmartBody can then be retrieved by right-clicking in the smartbody folder and choosing the 'SVN Update' option.

Linux

If you don't have SVN installed on your system, Ubuntu variants can run the following command as superuser:

```
apt-get install subversion
```

Then, run the following in the location where you wish to place SmartBody:

```
svn co https://smartbody.svn.sourceforge.net/svnroot/smartbody/trunk smartbody
```

Updates to SmartBody can then be retrieved by going into the smartbody directory and running:

```
svn update
```

Mac/OsX

Subversion is already installed on OsX platforms. Run the following in the location where you wish to place SmartBody:

```
svn co https://smartbody.svn.sourceforge.net/svnroot/smartbody/trunk smartbody
```

Updates to SmartBody can then be retrieved by going into the smartbody directory and running:

```
svn update
```

Building SmartBody

There are several SmartBody applications that can be built:

Application	Comment
sbm-fltk	a standalone SmartBody application and scene renderer
smartbody-dll	a dynamic library that can be incorporated into a game engine or other application
sbm-batch	a standalone SmartBody application that connects to smartbody-dll without a renderer
SbmDebuggerGui	(*Windows only*) a debugger/visualizer for running SmartBody processes
TtsRelayGui	(*Windows only*) incorporates any text-to-speech engine that uses the TtsRelay interface
FestivalRelay	Text-to-speech engine that uses Festival
MsSpeechRelay	(*Windows only*) Text-to-speech engine that uses Microsoft's built-in speech engine
OgreViewer	The Ogre rendering engine connected to SmartBody

In addition, there are several mobile applications that can be built:

Mobile Application	Comment
sbmjni	(*Android only*) Simple OpenGL ES front end for SmartBody for Android devices
sbm-ogre	(*Android only*) Ogre3D engine connected with SmartBody for Android devices
vh-wrapper	(*Android only*) SmartBody interface to Unity3D.
smartbody-openglES	(*iOS only*) Simple OpenGL ES front end for SmartBody for iOS devices
smartbody-ogre	(*iOS only*) Ogre3D engine connected with SmartBody for iOS devices
smartbody-unity	(*iOS only*) Unity3D project for SmartBody

Building SmartBody for Windows

You will need Visual Studio 2008 or higher to build SmartBody. In the top level SmartBody directory, Open the solution file 'vs2008.sln' and choose Build -> Build Solution. All libraries needed for the build have been included in the SmartBody repository.

You will need to install an ActiveMQ server if you wish to use the message system, although you can run SmartBody without it. The message system is used to send commands remotely to SmartBody. Download and install it from: http://activemq.apache.org/activemq-543-release.html

Building SmartBody for Linux

The Linux build requires a number of packages to be installed. For Ubuntu installations, the command apt-get can be used to retrieve and install those packages, for example:

```
sudo apt-get install cmake
```

Other Linux flavors can use rpm or similar installer.

You need to have the following packages installed:

Linux Packages Needed for SmartBody build
cmake
g++
liblog4cxx10-dev
libxerces-c3-dev
libgl1-mesa-dev
libglu1-mesa-dev
libglut-mesa-dev
xutils-dev
libxi-dev
freeglut3-dev
libglut3
libglew-dev
libxft-dev
libapr1-dev
libaprutil1-dev
libcppunit-dev
liblapack-dev
libblas-dev
build-essential
mono-devel
mono-xbuild
python-dev
libopenal-dev
libsndfile-dev
libalut-dev

Linux packages needed for text-to-speech engine
festival-dev

Linux packages needed for Ogre3D viewer
libzzip-dev
libxaw7-dev
libxxf86vm-dev

libxrandr-dev
libfreeimage-dev
nvidia-cg-toolkit
libois-dev

Linux packages needed for test suite
imagemagick

The Ogre-based renderer is not built by default. To build it, you will need to download the 1.6.5 source (www.ogre3d.org/download/source), build and install it into /usr/local. Next, uncomment the core/ogre-viewer directory from the core/CMakeLists.txt file. This will add the OgreViewer application to the build - the binary will be located in core/ogre-viewer/bin.

To build the Linux version:

1	Install the packages indicated above	

2	Download and install boost:
	
	Download Boost from: http://sourceforge.net/projects/boost/files/boost/1.44.0/boost_1_44_0.tar.gz/download
	
	Place in lib/ and unpack, build and install into /usr/local using:
	
	`./boostrap.sh`
	`./bjam --with-pythonsudo`
	`./bjam install`	

3	Download and install the Boost numeric bindings:
	
	Download from:
	http://mathema.tician.de/news.tiker.net/download/software/boost-numeric-bindings/boost-numeric-bindings-20081116.tar.gz
	
	Place in lib/ and unpack using:
	
	`tar -xvzf boost-numeric-bindings-20081116.tar.gz`
	`cd boost-numeric-bindingssudo`
	`sudo cp -R boost/numeric/bindings /usr/local/include/boost/numeric`	

4	Download and install FLTK 1.3.
	
	Download from: http://ftp.easysw.com/pub/fltk/1.3.0/fltk-1.3.0-source.tar.gz
	
	Place in lib/, unpack and configure:
	
	`./configure --enable-gl --disable-xinerama`
	`make`
	`sudo make install`	

5	Download and install ActiveMQ.
	
	Download from: http://www.apache.org/dyn/closer.cgi/activemq/activemq-cpp/source/activemq-cpp-library-3.4.0-src.tar.gz
	
	unpack to temp folder
	
	`./configure --disable-ssl`
	`make`
	`sudo make install`
	`sudo ldconfig`	

6	Get Open Dynamics Engine (ODE).
	
	Download ode-0.11.1 from:http://sourceforge.net/projects/opende/files/
	Place in lib/, unpack and configure:
	
	32 bits:
	
	`./configure --with-drawstuff=none`
	
	64 bits:
	
	`./configure --with-drawstuff=none --with-pic`
	
	`make`
	
	`sudo make install`	

7	(Optional) Build the elsender:
	
	`cd lib/elsender`
	`xbuild elsender.csproj  (ignore post-build error)`	

8	(Optional) Build the Ogre engine:
	
	Download the 1.6.5 version fromfrom: www.ogre3d.org/download/source
	
	`./configure`
	`sudo make install`	

9	Build speech tools and Festival that are located in the local SmartBody directory:
	
	`cd lib/festival/speech_tools`
	`./configure`
	`make install`
	`cd ../festival`
	`./configure`
	`make install`	

10	Make and install SmartBody:
	
	First, generate the Makefiles with cmake:
	
	`mkdir build`
	`cmake ..`
	
	Go to the build directory, make and install the applications
	
	`cd build`
	`make install`	

Building SmartBody for OSX

To build the OSX version:

1	Download and install boost:
	
	Download Boost from: http://sourceforge.net/projects/boost/files/boost/1.44.0/boost_1_44_0.tar.gz/download
	
	Place in lib/ and unpack, build and install into /usr/local using:
	
	`./boostrap.sh`
	`./bjam --with-pythonsudo`
	`sudo ./bjam install`	

2	Download and install the Boost numeric bindings:
	
	Download from:
	http://mathema.tician.de/news.tiker.net/download/software/boost-numeric-bindings/boost-numeric-bindings-20081116.tar.gz
	
	Place in lib/ and unpack using:
	
	`tar -xvzf boost-numeric-bindings-20081116.tar.gz`
	`cd boost-numeric-bindingssudo`
	`sudo cp -R boost/numeric/bindings /usr/local/include/boost/numeric`	

3	Download and install FLTK 1.3.
	
	Download from: http://ftp.easysw.com/pub/fltk/1.3.0/fltk-1.3.0-source.tar.gz
	
	Place in lib/, unpack and configure:
	
	`./configure --enable-gl --enable-shared --disable-xinerama`
	`make`
	`sudo make install`	

4 Download and install the ActiveMQ libraries.

Download from: http://www.apache.org/dyn/closer.cgi/activemq/activemq-cpp/source/activemq-cpp-library-3.4.0-src.tar.gz

Activemq-cpp will need to be changed slightly to work with OsX, as seen in this bug fix:

https://issues.apache.org/jira/browse/AMQCPP-369

Change the following in the activemq-cpp source code:

```
--- activemq/activemq-cpp/trunk/activemq-cpp/src/main/decaf/util/logging/Handler.h    2011/05/02 14:52:26    1098605
+++ activemq/activemq-cpp/trunk/activemq-cpp/src/main/decaf/util/logging/Handler.h    2011/05/02 14:52:30    1098606
@@ -49,9 +49,6 @@
    class DECAF_API Handler : public io::Closeable {
    private:


-       // Default Logging Level for Handler
-       static const Level DEFAULT_LEVEL;
-
        // Formats this Handlers output
        Formatter* formatter;


--- activemq/activemq-cpp/trunk/activemq-cpp/src/main/decaf/util/logging/Handler.cpp    2011/05/02 14:52:26    1098605
+++ activemq/activemq-cpp/trunk/activemq-cpp/src/main/decaf/util/logging/Handler.cpp    2011/05/02 14:52:30    1098606
@@ -28,11 +28,8 @@
using namespace decaf::util::logging;


////////////////////////////////////////////////////////////////////////////////
-const Level Handler::DEFAULT_LEVEL = Level::ALL;
-
-////////////////////////////////////////////////////////////////////////////////
Handler::Handler() : formatter(NULL), filter(NULL), errorManager(new ErrorManager()),
-                    level(DEFAULT_LEVEL), prefix("Handler") {
+                     level(Level::ALL), prefix("Handler") {
}


Then rebuild activemq-cpp using:

./configure --disable-ssl
make
sudo make install
sudo ldconfig
```

5	Get Open Dynamics Engine (ODE).
	
	Download ode-0.11.1 from:http://sourceforge.net/projects/opende/files/
	Place in lib/, unpack and configure:
	
	32 bits:
	
	`./configure --with-drawstuff=none`
	
	64 bits:
	
	`./configure --with-drawstuff=none --with-pic`
	
	`make`
	
	`sudo make install`	

6	Build and install xerces http://www.takeyellow.com/apachemirror//xerces/c/3/sources/xerces-c-3.1.1.tar.gz	
7	Build and install GLEW https://sourceforge.net/projects/glew/files/glew/1.6.0/glew-1.6.0.tgz/download	
8	Build and install log4cxx http://www.apache.org/dyn/closer.cgi/logging/log4cxx/0.10.0/apache-log4cxx-0.10.0.tar.gz	
9	Build and install OpenAL http://connect.creativelabs.com/openal/Downloads/openal-soft-1.13.tbz2	
10	Build and install FreeALUT http://connect.creativelabs.com/openal/Downloads/ALUT/freealut-1.1.0-src.zip	
11	Build and install libsndfile http://www.mega-nerd.com/libsndfile/files/libsndfile-1.0.25.tar.gz	

12	(Optional) Build the elsender:
	
	`cd lib/elsender`
	`xbuild elsender.csproj   (ignore post-build error)`	

13	(Optional) Build the Ogre engine:
	
	Download the 1.6.5 version fromfrom: www.ogre3d.org/download/source
	
	`./configure`
	`sudo make install`	

14	Build speech tools and Festival that are located in the local SmartBody directory:
	
	`cd lib/festival/speech_tools`
	`./configure`
	`make install`
	`cd ../festival`
	`./configure`
	`make install`	

15	Make and install SmartBody:
	
	First, generate the Makefiles with cmake:
	
	`mkdir build`
	`cmake ..`
	
	Go to the build directory, make and install the applications
	
	`cd build`
	`make install`	

Building SmartBody for Android

The SmartBody code is cross-compiled for the Android platform using the native development kit (NDK), which allows you to use gcc-like tools to build Android applications. This means that SmartBody is nearly fully-functional as a mobile application, since it uses the same code base as the desktop version of SmartBody. Many of the supporting libraries (such as ActiveMQ, boost, Xerces, Python, Festival, etc.) have already been built as static libraries and exist in the smartbody/android/lib directory.

Note that there are three different examples of Android applications using SmartBody that can be built:sbmjni, sbm-ogre, and vh-wrapper.

If your target hardware is the ARM architecture, some dependency libraries are already prebuilt at SmartBodyDir)/android/lib. Therefore you can build the SmartBody projects directly as below:

	Build Instructions for Android
1	Download and install android-sdk from : http://developer.android.com/sdk/index.html
2	Download and install the android-ndk from: http://developer.android.com/sdk/ndk/index.html (Note that when building on the windows platform, you will also need to install cygwin 1.7 or higher from http://www.cygwin.com/) Follow the installation instruction for both the SDK and the NDK. Be sure to set the correct path to android-sdk/bin, android-ndk/bin so the toolchain can be access correctly. For NDK, you also need to export the environment variable NDK_ROOT to the NDK installation directory (for example, export NDK_ROOT= "/path/to/ndk/directory")
3	Install Eclipse (http://www.eclipse.org/) and its Android ADT plug-in (http://developer.android.com/sdk/eclipse-adt.html) The supporting libraries have been built using Android version 2.3.3 or higher.
4	(Optional) Build sbm-jni sbmjni is a hello world project for SmartBody. It has very basic rendering and minimal functionality, but it helps demonstarte the SmartBody port on android. i) Go to (SmartBodyDir)/android/sbm-jni/ ii) ndk-build (Similar to gcc, you can set the option -j $number_threads to accelerate the build process with multi-threading). iii) Use Eclipse to open the project (SmartBodyDir)/android/sbm/. iv) Select Project->Build Project. Connect the device and then run the program as "Android Application".
5	(Optional) Build sbm-ogre sbm-ogre combines SmartBody and ogre for high quality rendering. Currently, it is very slow when rendering in deformable model mode. a. Go to (SmartBodyDir)/android/sbm-ogre/ b. ndk-build (Similar to gcc, you can set the option -j $number_threads to accelerate the build process with multi-threading). c. Use Eclipse to open the project (SmartBodyDir)/android/sbm-ogre/. d. Select Project->Build Project. ThConnect the device and then run the program as "Android Application".

6	(Optional) Build vh-wrapper vh-wrapper is a SmartBody interface to Unity3D. Note that SmartBody connects to Unity via Unity's native code plugin interface, which presently requires a Unity Pro license. In addition, the Unity project needs to be compiled for Android, so a Unity Android license is needed as well. a. Go to SmartBody/android/vh_wrapper/ b. ndk-build c. rename libvhwrapper.so to libvhwrapper-dll.so (for some reason, Android does not accept a build target name with "-") d. copy libvhwrapper-dll.so to the plug-in directory of Unity project. Build the Unity project for Android.

If you are targeting other hardware achitecture (x86, etc) or you prefer to rebuild all libraries from their sources, you will need to perform the following steps:

	Building Supporting Libraries
1	Building Boost for Android: Download BOOST library, extract it into SmartBody/lib modify libs\filesystem\v2\src\v2_operations.cpp, change: # if !defined(__APPLE__) && !defined(__OpenBSD__) # include <sys/statvfs.h> # define BOOST_STATVFS statvfs # define BOOST_STATVFS_F_FRSIZE vfs.f_frsize # else #ifdef __OpenBSD__ # include <sys/param.h> #endif to: # if !defined(__APPLE__) && !defined(__OpenBSD__) && !defined(__ANDROID__) # include <sys/statvfs.h> # define BOOST_STATVFS statvfs # define BOOST_STATVFS_F_FRSIZE vfs.f_frsize # else #ifdef __OpenBSD__ # include <sys/param.h> #elif defined(__ANDROID__) # include <sys/vfs.h> #endif modifiy the file SmartBody/android/boost/userconfig.jam, look for : ANDROID_NDK = ../android-ndk ; and change the directory "../android-ndk" so it points to the android NDK directory You may also need to change all arm-linux-androideabi-xxx to the corresponding toolchain name based on your target architecture and platform. (use Cygwin in Windows platform) ./bootstrap.sh ./bjam --without-python --without-math --without-mpi --without- iostreams toolset=gcc-android4.4.3 link=static runtime-link=static target-os=linux --stagedir=android stage
2	Building iconv TODO
3	Building xerces TODO

4	Building clapack TODO

Building SmartBody for iOS

The iOS build requires two steps: building libraries via the command console, then building applications and libraries via XCode.

	Compiling using console
1	Cross compiling apr http://archive.apache.org/dist/apr/ Download apr-1.3.*, copy setup-iphoneos.sh and setup-iphonesimulator.sh from trunk/ios/activemq/apr to that folder Change the SBROOT inside setup-iphoneos.sh and setup-iphonesimulator.sh to your trunk directory Run both of the scripts Go to trunk/ios/activemq/apr/iphone*/include ,edit line 79 of apr_general.h to be: #if defined(CRAY) || (defined(__arm) && !(defined(LINUX) || defined(__APPLE__)))
2	Cross compiling apr-util http://archive.apache.org/dist/apr/ Download apr-util-1.3.*, copy setup-iphoneos.sh and setup-iphonesimulator.sh from trunk/ios/activemq/apr-util to that folder Change the SBROOT inside setup-iphoneos.sh and setup-iphonesimulator.sh to your trunk directory Run both of the scripts
3	Cross compiling activemq-cpp-library http://apache.osuosl.org/activemq/activemq-cpp/source/ Download activemq-cpp-library-3.4.0, copy setup-iphoneos.sh and setup-iphonesimulator.sh from trunk/ios/activemq/apr-util to that folder change src/main/decaf/lang/system.cpp line 471 inside activemq folder from "#if defined (__APPLE__)" to "#if 0" Change the SBROOT inside setup-iphoneos.sh and setup-iphonesimulator.sh to your trunk directory Run both of the scripts Note: for smartbody iphone running on unity, we need to rename variables inside activemq-cpp-library decaf/internal/util/zip/*.c to avoid conflict symbols. If you don't want to do that, you can directly use the one under trunk/ios/activemq/activemq-cpp/libs/activemq-unity
4	Cross compiling xerces-c http://xerces.apache.org/xerces-c/download.cgi Download xerces-c-3.1.1.tar.gz, unzip it, copy setup-iphoneos.sh and setup-iphonesimulator.sh from trunk/ios/activemq/activemq-cpp to that folder Change the SBROOT inside setup-iphoneos.sh and setup-iphonesimulator.sh to your trunk directory Run both of the scripts
5	Cross compiling ODE http://sourceforge.net/projects/opende/files/ Download ode-0.11.1.zip, copy setup-iphoneos.sh and setup-iphonesimulator.sh from trunk/ios/activemq/activemq-cpp to that folder Change the SBROOT inside setup-iphoneos.sh and setup-iphonesimulator.sh to your trunk directory Run both of the scripts

6	Cross compiling clapack (Optional since you can add Acceleration framework from xcode project) http://www.netlib.org/clapack/ Download clapack-3.2.1-CMAKE.tgz, unzip and copy toolchain-iphone*.cmake and setup-iphone*.sh from trunk/ios/activemq/activemq-cpp to that folder To make a Unity compatible build, you need to modify the cmake file. If not used for Unity Iphone, you can skip the following step: -Go to clapack/CMakeLists.txt, comment out include(CTest) and add_subdirectory(TESTING) -Go to clapack/BLAS/CMakeLists.txt, comment out add_subdirectory(TESTING) -Go to clapack/F2CLIBS/libf2c/CMakeLists.txt, take out main.c from first SET so it became set(MISC f77vers.c i77vers.c s_rnge.c abort_.c exit_.c getarg_.c iargc_.c getenv_.c signal_.c s_stop.c s_paus.c system_.c cabs.c ctype.c derf_.c derfc_.c erf_.c erfc_.c sig_die.c uninit.c) -Go to clapack/SRC/CMakeLists.txt, take out ../INSTALL/lsame.c from first SET so it became set(ALLAUX maxloc.c ilaenv.c ieeeck.c lsamen.c iparmq.c ilaprec.c ilatrans.c ilauplo.c iladiag.c chla_transtype.c ../INSTALL/ilaver.c) # xerbla.c xerbla_array.c After doing this, change the SBROOT inside setup-iphoneos.sh and setup-iphonesimulator.sh, run the scripts according to what you are building to.
7	Cross compiling python Go to trunk/ios/python, modify the SBROOT inside setup-iphoneos.sh, then run the script.
8	(Optional) Cross compiling pocket sphinx http://www.rajeevan.co.uk/pocketsphinx_in_iphone/ The steps are on the website. Since the results are not that good on Unity and I don't have time fully test out, the code is not intergrated into smartbody yet. When integrating pocketsphinx with Unity, it would have duplicated symbol problem(this might be the reason of bad recognizing result, it's under sphinx/src/util). I already built a library for Unity that can be used directly. Also for Unity you may need get prime31 iphone plugin AudioRecorder

	Compiling using Xcode4
1	Build bonebus Open smartbody-iphone.xcworkspace, select the scheme to be bonebus, build
2	Build boost http://www.boost.org/users/history/version_1_44_0.html Download boost_1_44_0.tar.gz, unzip to trunk/ios/boost. Make sure the folder name is boost_1_44_0. Open smartbody-iphone.xcworkspace, select boost_system, boost_filesystem, boost_regex, build them seperately. http://mathema.tician.de/news.tiker.net/download/software/boost-numeric-bindings/boost-numeric-bindings-20081116.tar.gz Download boost_numeric_bindings, unzip it to trunk/ios/boost, make sure the name is boost_numeric_bindings
3	Build steersuite Open smartbody-iphone.xcworkspace, select the steerlib, pprAI, build them seperately.
4	Build vhmsg Open smartbody-iphone.xcworkspace, select scheme vhmsg and build.4
5	Build vhcl Since the vhcl_log.cpp hasn't been changed from VH group, you have to copy trunk/ios/vhcl/vhcl_log.cpp to trunk/lib/vhcl/src/vhcl_log.cpp for now. Open smartbody-iphone.xcworkspace, select scheme vhcl and build.
6	Build wsp Open smartbody-iphone.xcworkspace, select scheme wsp and build.
7	Build smartbody-lib Open smartbody-iphone.xcworkspace, select scheme smartbody-lib and build.

8	(Optional) Build smartbody-dll Open smartbody-iphone.xcworkspace, select scheme smartbody-dll and build.
9	(Optional) Build vhwrapper-dll (for Unity only) Open smartbody-iphone.xcworkspace, select scheme vhwrapper-dll and build.

Once those steps have been completed, you can build any of three applications:

smartbody-openglES - a simple example of using SmartBody with OpenGL

smartbody-ogre - an example of using SmartBody with the Ogre3D rendering engine

smartbody-unity - The Unity3D game engine connected to SmartBody.

Make sure that your iOS device is connected and follow any of the three applications below:

	Building smartbody-openglES
1	Build smartbody-openglES Go to trunk/ios/applications/minimal, open smartbody-iphone.xcodeproj, build and run. Note: Under smartbody-openglES project Frameworks, you should see all the libraries existing. If not, go over previous steps to check if anything is wrong

	Building smartbody-ogre
1	Build smartbody-ogre Download OgreSDK: http://www.ogre3d.org/download/sdk Build the Ogre iPhone libraries as indicated here: http://www.ogre3d.org/tikiwiki/tiki-index.php?page=Building%20From%20Source%20-%20iPhone&redirectpage=Building%20From%20Sour Make sure all the libraries exist inside build/lib/Debug
2	Go to trunk/ios/applications/ogreIphone, open smartbody-ogre.xcodeproj go to smartbody-ogre project, set OGRE_SDK_ROOT to your ogreSDK directory, set OGRE_SRC_ROOT to your ogre source directory. Select scheme smartbody-ogre, build and run. If the program hangs on boost thread function, try rebuild the ogre iphone dependencies boost libs (pthread, date_time), alternative way is ogre iOS libraries with Boost symbol turned off which may affect the results. Note: ogre 1.8 seems to have trouble when building for iphone/ipad, use ogre 1.7.3.It is extremely slow running on armv6 ipod(after testing), something wrong with the texture and shader. So maybe should just run on armv7 iPhone/iPad

	Building smartbody-unity
1	TODO