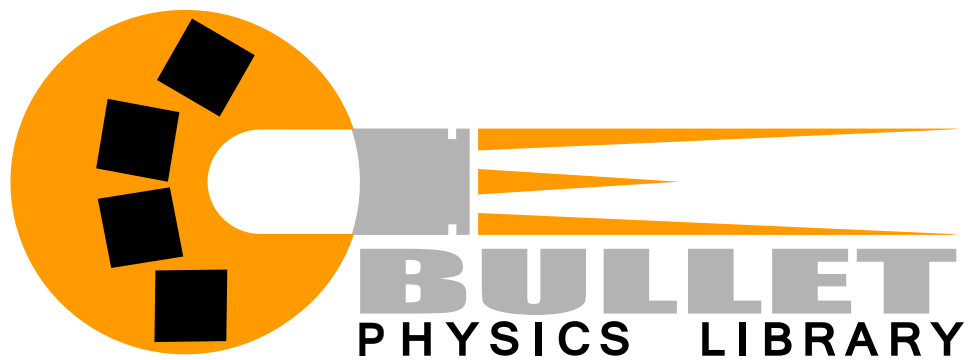


Bullet 2.81 Quickstart Guide

Erwin Coumans

October 8, 2012



Contents

1	Introduction to Bullet	2
1.1	Main Features	2
1.2	Contact and Support	2
1.3	What's new	3
1.3.1	New in Bullet 2.81	3
2	Building the Bullet SDK and demos	4
2.1	Microsoft Visual Studio	4
2.2	Using Premake	4
2.2.1	Premake Visual Studio project generation	4
2.2.2	Premake Mac OSX Xcode project generation	5
2.2.3	Premake iOS Xcode project generation	5
2.2.4	Premake GNU Makefile generation	5
2.3	Using cmake	5
2.4	Using autotools	6
3	Hello World	7
3.1	C++ console program	7
4	Frequently asked questions	9
	Source code listings	10
	Index	11

1 Introduction to Bullet

Bullet Physics is a professional open source collision detection, rigid body and soft body dynamics library. The library is free for commercial use under the [zlib license](#).

1.1 Main Features

- Open source C++ code under zlib license and free for any commercial use on all platforms including PLAYSTATION 3, XBox 360, Wii, PC, Linux, Mac OSX, Android and iPhone
- Discrete and continuous collision detection including ray and convex sweep test. Collision shapes include concave and convex meshes and all basic primitives
- Fast and stable rigid body dynamics constraint solver, vehicle dynamics, character controller and slider, hinge, generic 6DOF and cone twist constraint for ragdolls
- Soft Body dynamics for cloth, rope and deformable volumes with two-way interaction with rigid bodies, including constraint support
- Maya Dynamica plugin, Blender integration, COLLADA physics import/export support

1.2 Contact and Support

- Public forum for support and feedback is available at <http://bulletphysics.org>

- PLAYSTATION 3 licensed developers can download an optimized version for Cell SPU through Sony [PS3 Devnet](#).

1.3 What's new

1.3.1 New in Bullet 2.81

- SIMD and Neon optimizations for iOS and Mac OSX, thanks to a contribution from Apple
- Rolling Friction using a constraint, thanks to Erin Catto for the idea. See `Demos/RollingFrictionDemo/RollingFrictionDemo.cpp`
- XML serialization. See `Bullet/Demos/BulletXmlImportDemo` and `Bullet/Demos/SerializeDemo`
- Gear constraint. See `Bullet/Demos/ConstraintDemo`.
- Improved continuous collision response, feeding speculative contacts to the constraint solver. See `Bullet/Demos/CcdPhysicsDemo`
- Improved premake4 build system including support for Mac OSX, Linux and iOS
- Refactoring of collision detection pipeline using stack allocation instead of modifying the collision object. This will allow better future multithreading optimizations.

2 Building the Bullet SDK and demos

Windows developers can download the zipped sources of Bullet from <http://bullet.googlecode.com>. Mac OS X, Linux and other developers should download the gzipped tar archive. Bullet provides several build systems.

2.1 Microsoft Visual Studio

After unzipping the source code you can open the `Bullet/build/vs2010/0BulletSolution.sln`, hit F5 and your first Bullet demo will run. Note that by default Visual Studio uses an unoptimized Debug configuration that is very slow. It is best to enable the Release configuration.

2.2 Using Premake

Premake is a meta build system based on the Lua scripting language that can generate project files for Microsoft Visual Studio, Apple Xcode as well as Makefiles for GNU make and other build systems. Bullet comes with Premake executables for Windows, Mac OSX and Linux.

2.2.1 Premake Visual Studio project generation

You can double-click on `Bullet/build/vs2010.bat` to generate Visual Studio 2010 project files and solution. This batch file calls Premake. Just open `Bullet/build/vs2010/0BulletSolution.sln`

2.2.2 Premake Mac OSX Xcode project generation

On Mac OSX it is easiest to open a Terminal window and switch current directory to Bullet/build and use the following command to generate XCode projects:

Source Code 2.1: Premake for Mac OSX

```
1 cd Bullet/build
2 ./premake_osx xcode4
3 open xcode4/0BulletSolution.xcworkspace
```

2.2.3 Premake iOS Xcode project generation

XCode project generation for iOS, such as iPhone and iPad, is similar to Mac OSX. Just provide the `-ios` option to premake and premake will automatically customize the project and append ios to the directory:

Source Code 2.2: Premake for iOS

```
1 cd Bullet/build
2 ./premake_osx --ios xcode4
3 open xcode4ios/0BulletSolution.xcworkspace
```

Note that Bullet comes with a modified version of `premake_osx` that enables the iOS specific customizations that are required by XCode.

2.2.4 Premake GNU Makefile generation

You can also generate GNU Makefiles for Mac OSX or Linux using premake:

Source Code 2.3: Premake to GNU Makefile

```
1 cd Bullet/build
2 ./premake_osx gmake
3 cd gmake
4 make config=release64
```

2.3 Using cmake

Similar to premake, CMake adds support for many other build environments and platforms, including Microsoft Visual Studio, XCode for Mac OSX, KDevelop for Linux and Unix Makefiles. Download and install CMake from <http://cmake.org> and use the CMake `cmake-gui` tool.

2.4 Using autotools

Open a shell terminal and go to the Bullet root directory. Then execute

Source Code 2.4: autotools to Makefile

```
1 ./autogen.sh
2 ./configure
3 make
```

The `autogen.sh` step is optional and not needed for downloaded packages.

3 Hello World

3.1 C++ console program

Let's discuss the creation of a basic Bullet simulation from the beginning to the end. For simplicity we print the state of the simulation to console using `printf`, instead of using 3D graphics to display the objects. The source code of this tutorial is located in `Demos/HelloWorld/HelloWorld.cpp`.

It is a good idea to try to compile, link and run this `HelloWorld.cpp` program first.

As you can see in 3.1 you can include a convenience header file `btBulletDynamicsCommon.h`.

Source Code 3.1: `HelloWorld.cpp` include header

```
16 #include "btBulletDynamicsCommon.h"
17 #include <stdio.h>
18
19 /// This is a Hello World program for running a basic Bullet physics simulation
20
21 int main(int argc, char** argv)
22 {
```

Now we create the dynamics world:

Source Code 3.2: `HelloWorld.cpp` initialize world

```
27
28 ///collision configuration contains default setup for memory, collision setup. Advanced users can
   create their own configuration.
29 btDefaultCollisionConfiguration* collisionConfiguration = new btDefaultCollisionConfiguration();
30
31 ///use the default collision dispatcher. For parallel processing you can use a different dispatcher
   (see Extras/BulletMultiThreaded)
32 btCollisionDispatcher* dispatcher = new btCollisionDispatcher(collisionConfiguration);
33
34 ///btDbvtBroadphase is a good general purpose broadphase. You can also try out btAxis3Sweep.
35 btBroadphaseInterface* overlappingPairCache = new btDbvtBroadphase();
36
37 ///the default constraint solver. For parallel processing you can use a different solver (see
   Extras/BulletMultiThreaded)
38 btSequentialImpulseConstraintSolver* solver = new btSequentialImpulseConstraintSolver;
39
40 btDiscreteDynamicsWorld* dynamicsWorld = new btDiscreteDynamicsWorld(dispatcher,
   overlappingPairCache, solver, collisionConfiguration);
41
42 dynamicsWorld->setGravity(btVector3(0,-10,0));
```


3 Hello World

Once the world is created you can step the simulation as follows:

Source Code 3.3: HelloWorld.cpp step simulation

```
115 for (i=0;i<100;i++)
116 {
117     dynamicsWorld->stepSimulation(1.f/60.f,10);
118
119     //print positions of all objects
120     for (int j=dynamicsWorld->getNumCollisionObjects()-1; j>=0 ;j--)
121     {
122         btCollisionObject* obj = dynamicsWorld->getCollisionObjectArray()[j];
123         btRigidBody* body = btRigidBody::upcast(obj);
124         if (body && body->getMotionState())
125         {
126             btTransform trans;
127             body->getMotionState()->getWorldTransform(trans);
128             printf("world_pos=%f,%f,%f\n",float(trans.getOrigin().getX()),float(trans.getOrigin().getY
129                 ()),float(trans.getOrigin().getZ()));
130         }
131     }
```

At the end of the program you delete all objects in the reverse order of creation. Here is the cleanup listing of our HelloWorld.cpp program.

Source Code 3.4: HelloWorld.cpp cleanup

```
138 //remove the rigidbodies from the dynamics world and delete them
139 for (i=dynamicsWorld->getNumCollisionObjects()-1; i>=0 ;i--)
140 {
141     btCollisionObject* obj = dynamicsWorld->getCollisionObjectArray()[i];
142     btRigidBody* body = btRigidBody::upcast(obj);
143     if (body && body->getMotionState())
144     {
145         delete body->getMotionState();
146     }
147     dynamicsWorld->removeCollisionObject( obj );
148     delete obj;
149 }
150
151 //delete collision shapes
152 for (int j=0;j<collisionShapes.size();j++)
153 {
154     btCollisionShape* shape = collisionShapes[j];
155     collisionShapes[j] = 0;
156     delete shape;
157 }
158
159 //delete dynamics world
160 delete dynamicsWorld;
161
162 //delete solver
163 delete solver;
164
165 //delete broadphase
166 delete overlappingPairCache;
167
168 //delete dispatcher
169 delete dispatcher;
170
171 delete collisionConfiguration;
172
173 //next line is optional: it will be cleared by the destructor when the array goes out of scope
174 collisionShapes.clear();
175
```

4 Frequently asked questions

Here is a placeholder for a future FAQ. For more information it is best to visit the Bullet Physics forums and wiki at <http://bulletphysics.org>.

Source Code Listings

2.1	Premake for Mac OSX	5
2.2	Premake for iOS	5
2.3	Premake to GNU Makefile	5
2.4	autotools to Makefile	6
3.1	HelloWorld.cpp include header	7
3.2	HelloWorld.cpp initialize world	7
3.3	HelloWorld.cpp step simulation	8
3.4	HelloWorld.cpp cleanup	8

Index

CMake, 5

premake, 4

zlib license, 2