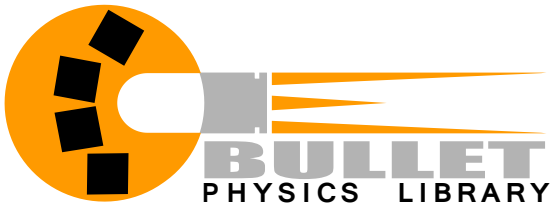
Bullet 2.83 Quickstart Guide

Erwin Coumans

November 17, 2018



Contents

Chapter 1

Introduction to Bullet

Bullet Physics is a professional open source collision detection, rigid body and soft body dynamics library. Bullet Physics targets real-time and interactive use in games, visual effects in movies and robotics. 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
- Native binary .bullet file format and example importers for URDF, Wavefront obj and Quake bsp files.

1.2 Contact and Support

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

1.3 What's new

1.3.1 New in Bullet 2.83

- New ExampleBrowser, replacing the standalone demos. Not all demos have been ported over yet, it will happen in upcoming releases. It is recommended to use an OpenGL 3+ system, although there is limited OpenGL 2 fallback. See `examples/ExampleBrowser`.
- Import of Universal Robot Description Files (URDF). See `examples/Importers/ImportURDFDemo`. You can use File-Open in the ExampleBrowser.
- Improved support for btMultiBody with multi-degree of freedom mobilizers, thanks to Jakub Stepien. See `examples/MultiBody/MultiDofDemo`.
- New btGeneric6DofSpring2Constraint, replacing the old generic 6dof constraint, thanks to Gabor PUHR and Tamas Umenhoffer. See `examples/Dof6Spring2Setup`
- OpenCL demo integrated in the ExampleBrowser. The demo is disabled by default, because it only works on high-end desktop GPU systems with compatible up-to-date OpenCL 1.1+ driver. Use $--enable\_experimental\_opencl$ command-line argument in the ExampleBrowser.

1.3.2 New in Bullet 2.82

- Featherstone articulated body algorithm implementation with integration in the Bullet constraint solver. See `examples/MultiBodyDemo`
- New MLCP constraint solver interface for higher quality direct solvers. Dantzig (OpenDE), PATH and Projected Gauss Seidel MLCP solvers, with fallback to the original Bullet sequential impulse solver. See `src/BulletDynamics/MLCPSolvers`
- New btFixedConstraint as alternative to a btGeneric6DofConstraint with all DOFs locked. See `Demos/VoronoiFractureDemo`
- Various bug fixes, related to force feedback and friction. Improved performance between btCompoundShape using the new btCompoundCompoundCollisionAlgorithm. See the commit log at https://code.google.com/p/bullet/source/list

1.3.3 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.

Chapter 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 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.

2.1.1 Premake Visual Studio project generation

You can double-click on `Bullet/vs2010.bat` to generate Visual Studio 2010 project files and solution. This batch file calls Premake. Just open `Bullet/build3/vs2010/0_Bullet3Solution.sln`. Newer versions of Visual Studio should automatically convert from the 2010 solution.

2.1.2 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

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

2.1.3 Premake GNU Makefile generation

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

Source Code 2.2: Premake to GNU Makefile

```
cd Bullet/build
./premake_osx gmake
cd gmake
make config=release64
```

2.2 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. Alternatively use a terminal, change to the Bullet root directory and use for example the following commands:

Source Code 2.3: CMake to GNU Makefile

```
cmake .
make
```

An example to create an XCode project on Mac OSX using CMake:

Source Code 2.4: CMake to Xcode

```
mkdir xcode
cd xcode
cmake .. -G Xcode
open BULLET_PHYSICS.xcodeproj
```

2.3 Executing the Example Browser

After building the SDK, there are some binary executables in the bin folder. You can execute the $bin/App_ExampleBrowser_*$ to experiment with the Bullet Physics SDK.

Chapter 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 `examples/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`.

While linking to Bullet Physics make sure the link order is: BulletSoftBody, BulletDynamics, BulletCollision and LinearMath.

Source Code 3.1: HelloWorld.cpp include header

```
17 #include "btBulletDynamicsCommon.h"
18 #include <stdio.h>
19
20 /// This is a Hello World program for running a basic Bullet physics simulation
21
22 int main(int argc, char** argv)
23 {
```

Now we create the dynamics world:

Source Code 3.2: HelloWorld.cpp initialize world

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

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 < 150; 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       btTransform trans;
125       if (body && body->getMotionState())
126       {
127         body->getMotionState()->getWorldTransform(trans);
128       }
129       else
130       {
131         trans = obj->getWorldTransform();
132       }
133       printf("world␣pos␣object␣%d␣=␣%f,%f,%f\n", j, float(trans.getOrigin().getX()), float
          (trans.getOrigin().getY()), float(trans.getOrigin().getZ()));
134     }
135   }
```

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

```
142
143   //remove the rigidbodies from the dynamics world and delete them
144   for (i = dynamicsWorld->getNumCollisionObjects() - 1; i >= 0; i--)
145   {
146     btCollisionObject* obj = dynamicsWorld->getCollisionObjectArray()[i];
147     btRigidBody* body = btRigidBody::upcast(obj);
148     if (body && body->getMotionState())
149     {
150       delete body->getMotionState();
151     }
152     dynamicsWorld->removeCollisionObject(obj);
153     delete obj;
154   }
155
156   //delete collision shapes
157   for (int j = 0; j < collisionShapes.size(); j++)
158   {
159     btCollisionShape* shape = collisionShapes[j];
160     collisionShapes[j] = 0;
161     delete shape;
162   }
163
164   //delete dynamics world
165   delete dynamicsWorld;
166
167   //delete solver
168   delete solver;
169
170   //delete broadphase
171   delete overlappingPairCache;
172
173   //delete dispatcher
```

```
174   delete dispatcher;
175
176   delete collisionConfiguration;
177
178   //next line is optional: it will be cleared by the destructor when the array goes out of
          scope
179   collisionShapes.clear();
180 }
```

Chapter 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

Index