Bullet 2.81 Quickstart Guide

Erwin Coumans

October 6, 2012

Contents

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.

1.4 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.

1.4.1 Using premake with Visual Studio

After unzipping the source code, you can open the `Bullet/build` directory and double click on `vs2010.bat` to generate Visual Studio 2010 project files and solution. Just open `Bullet/build/vs2010/0BulletSolution.sln`

1.4.2 Using premake with Xcode for Mac OSX or iOS

1.4.3 Using cmake

1.4.4 Using autotools

2 Hello World

2.1 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 2.1 you can include a convenience header file `btBulletDynamicsCommon.h`.

Source Code 2.1: HelloWorld.cpp include header

```cpp
#include "btBulletDynamicsCommon.h"
#include <stdio.h>

/// This is a Hello World program for running a basic Bullet physics simulation

int main(int argc, char** argv)
{
```

Now we create the dynamics world:

Source Code 2.2: HelloWorld.cpp initialize world

```cpp

	///collision configuration contains default setup for memory, collision setup. Advanced users can create their own configuration.
	btDefaultCollisionConfiguration* collisionConfiguration = new btDefaultCollisionConfiguration();

	///use the default collision dispatcher. For parallel processing you can use a diffent dispatcher (see Extras/BulletMultiThreaded)
	btCollisionDispatcher* dispatcher = new btCollisionDispatcher(collisionConfiguration);

	///btDbvtBroadphase is a good general purpose broadphase. You can also try out btAxis3Sweep.
	btBroadphaseInterface* overlappingPairCache = new btDbvtBroadphase();

	///the default constraint solver. For parallel processing you can use a different solver (see Extras/BulletMultiThreaded)
	btSequentialImpulseConstraintSolver* solver = new btSequentialImpulseConstraintSolver;

	btDiscreteDynamicsWorld* dynamicsWorld = new btDiscreteDynamicsWorld(dispatcher,overlappingPairCache,solver,collisionConfiguration);

	dynamicsWorld->setGravity(btVector3(0,-10,0));
```

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

Source Code 2.3: HelloWorld.cpp step simulation

```
  for (i=0;i<100;i++)
  {
    dynamicsWorld->stepSimulation(1.f/60.f,10);

    //print positions of all objects
    for (int j=dynamicsWorld->getNumCollisionObjects()-1; j>=0 ;j--)
    {
      btCollisionObject* obj = dynamicsWorld->getCollisionObjectArray()[j];
      btRigidBody* body = btRigidBody::upcast(obj);
      if (body && body->getMotionState())
      {
        btTransform trans;
        body->getMotionState()->getWorldTransform(trans);
        printf("world␣pos␣=␣%f,%f,%f\n",float(trans.getOrigin().getX()),float(trans.getOrigin().getY()),float(trans.getOrigin().getZ()));
      }
    }
  }
```

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 2.4: HelloWorld.cpp cleanup

```

  //remove the rigidbodies from the dynamics world and delete them
  for (i=dynamicsWorld->getNumCollisionObjects()-1; i>=0 ;i--)
  {
    btCollisionObject* obj = dynamicsWorld->getCollisionObjectArray()[i];
    btRigidBody* body = btRigidBody::upcast(obj);
    if (body && body->getMotionState())
    {
      delete body->getMotionState();
    }
    dynamicsWorld->removeCollisionObject( obj );
    delete obj;
  }

  //delete collision shapes
  for (int j=0;j<collisionShapes.size();j++)
  {
    btCollisionShape* shape = collisionShapes[j];
    collisionShapes[j] = 0;
    delete shape;
  }

  //delete dynamics world
  delete dynamicsWorld;

  //delete solver
  delete solver;

  //delete broadphase
  delete overlappingPairCache;

  //delete dispatcher
  delete dispatcher;

  delete collisionConfiguration;

  //next line is optional: it will be cleared by the destructor when the array goes out of scope
  collisionShapes.clear();
```

3 Frequently asked questions

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

3.1 Build problems

todo

3.2 Performance issues

todo

3.3 Physics issues

todo

3.4 Collision issues

todo

3.5 Ray testing

todo

Source Code Listings

Index