Intro to Bullet Physics

Ian Parberry

From the User Manual

"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."

Main Features

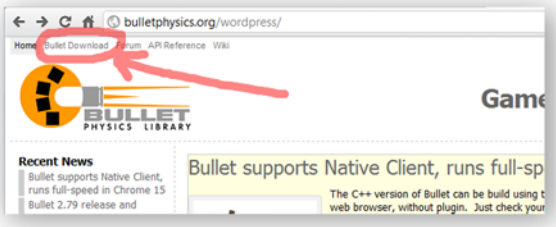
- Open source C++ code under Zlib license and free for any commercial use on all platforms including PLAYSTATION 3, XBox 360, Wii, PC, Linux, Mac OSX and iPhone
- Discrete and continuous collision detection including ray and convex sweep test. Collision shapes include concave and convex meshes and all basic primitives

Main Features

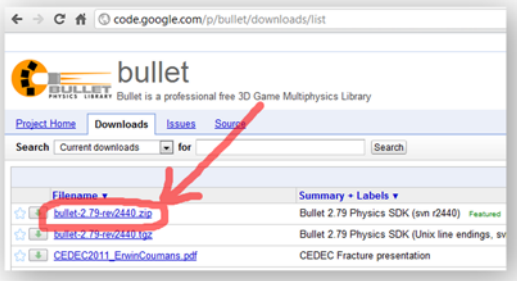
- Fast and stable rigid body dynamics constraint solver, vehicle dynamica, character controller and slider, hinge, generic 6DOF and cone twist constraint for ragdolls
- Soft Body dynamics for cloth, rope and deformable volumes with two-way interaction with rigid bodies, including constraint support
- Maya Dynamica plugin, Blender integration, COLLADA physics import/export support

Where is it Used?

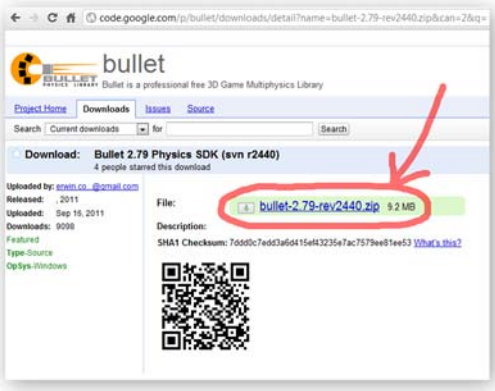
http://bulletphysics.org



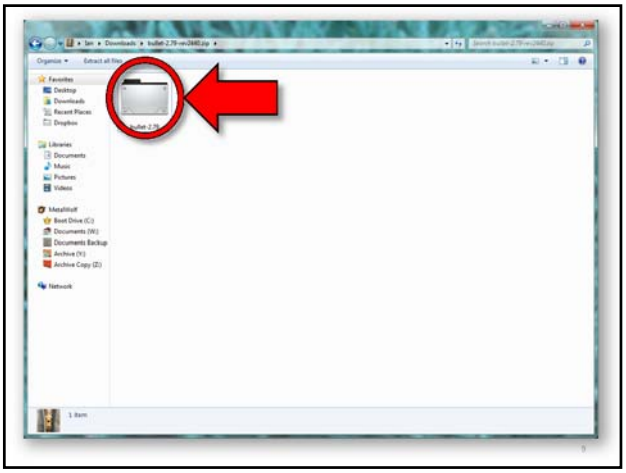
Download the Zip File



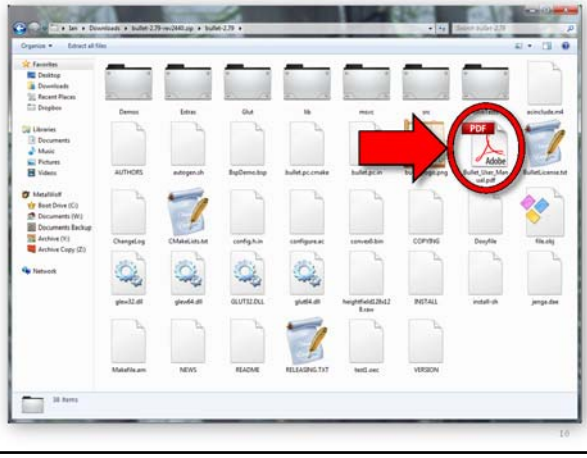
7



8



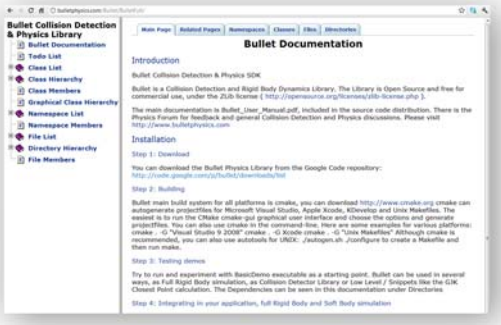
9



10

11

http://bulletphysics.com /Bullet/BulletFull/



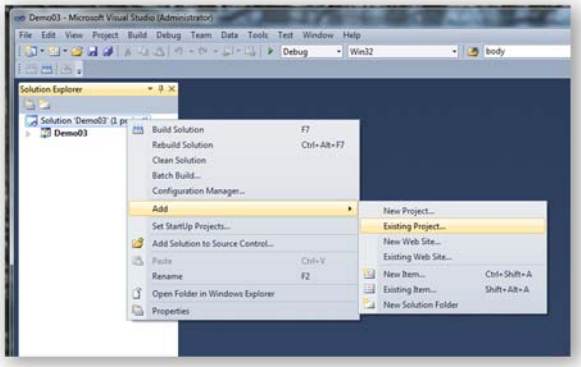
12

Adding Bullet Physics to Your Code

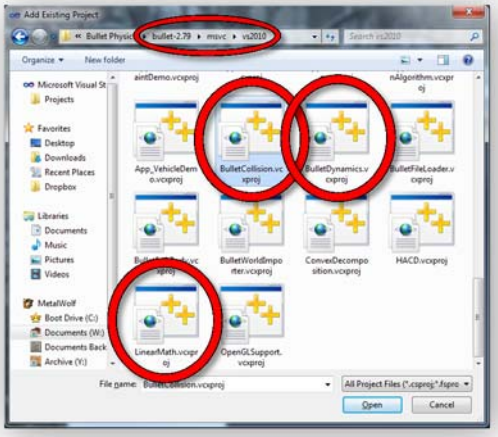
Add these Projects to your Solution:

`BulletDynamics.vcxproj`
`BulletCollision.vcxproj`
`LinearMath.vcxproj`

13



14



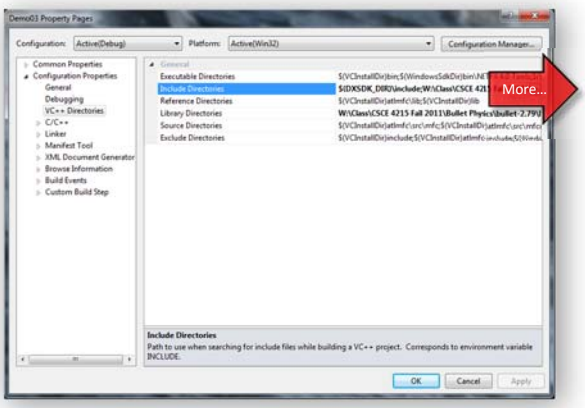
15

Adding Bullet Physics to your Code

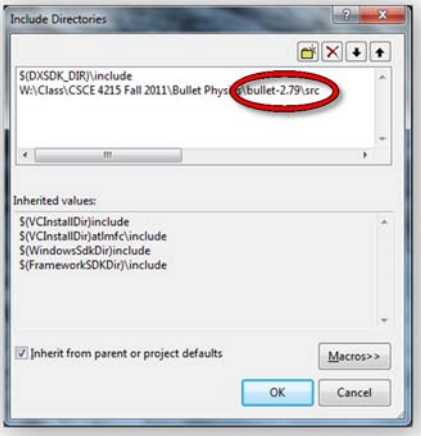
- Add the include path: `Bullet-2.79\src`
- Add the library path: `Bullet-2.79\lib`
- Add to your source file the line

```
#include "btBulletDynamicsCommon.h"
```

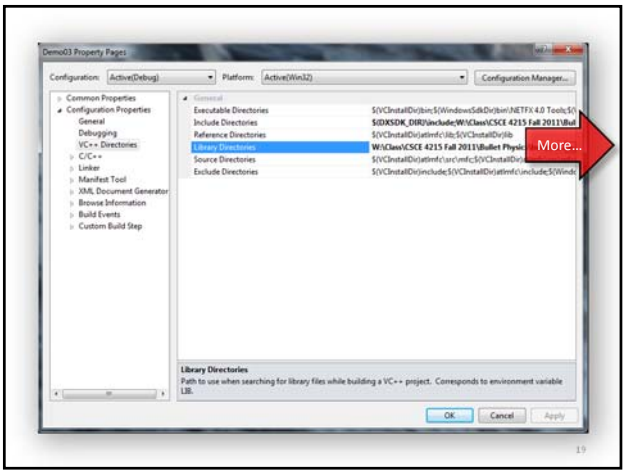
16

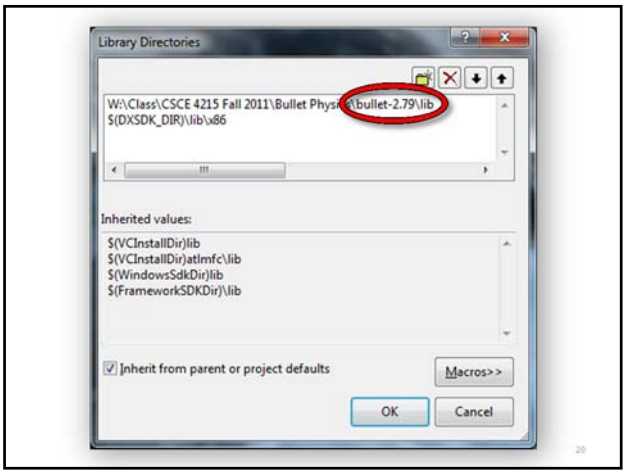


17



18





Adding Bullet Physics to Your Code

- Check out `CcdPhysicsDemo` how to create a `btDiscreteDynamicsWorld`, `btCollisionShape`, `btMotionState` and `btRigidBody`.
- Each frame call the `stepSimulation` on the dynamics world, and synchronize the world transform for your graphics object.

21

Adding Bullet Physics to Your Code

Basic order of operations:

- Create Core Objects
- Physics Loop Stuff (repeat until done)
- Clean up

22

Core Objects

- `btDefaultCollisionConfiguration`
- `btCollisionDispatcher`
- `btBroadphaseInterface`
- `btSequentialImpulseConstraintSolver`
- `btDiscreteDynamicsWorld`

23

Physics Loop Stuff

1. Add rigid bodies, soft bodies, etc.
2. Modify physics parameters
3. Step simulation
4. Return to Step 1, until end of Physics Loop.

24

Cleanup

1. Delete objects created in Physics Loop
2. Clear various arrays
3. Delete core objects

25

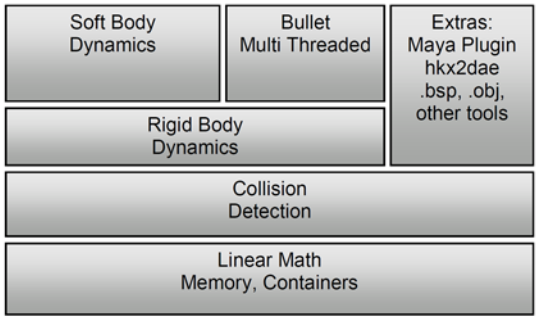
Software Design

Bullet has been designed to be customizable and modular. The developer can, for example,

- use only the collision detection component
- use the rigid body dynamics component without soft body dynamics component
- use only small parts of a the library and extend the library in many ways

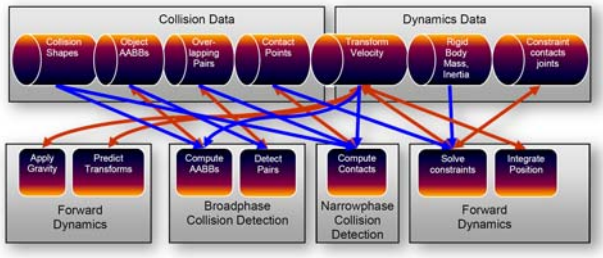
26

Components



27

Rigid Body Physics Pipeline



28

Dynamics World

A *dynamics world* provides a high level interface that manages your physics objects and constraints, and implements the update of all objects each frame.

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

Next: How the parameters were created.

29

collisionConfiguration and dispatcher

```
//default setup for memory, collision
btDefaultCollisionConfiguration*
 collisionConfiguration = new
  btDefaultCollisionConfiguration();
//use default collision dispatcher
btCollisionDispatcher*
 dispatcher = new
   btCollisionDispatcher(
    collisionConfiguration);
```

30

overlappingPairCache and solver

```
//btDbvtBroadphase is a good general
//purpose broadphase
btBroadphaseInterface*
 overlappingPairCache = new
  btDbvtBroadphase();
//the default constraint solver
btSequentialImpulseConstraintSolver*
 solver = new
  btSequentialImpulseConstraintSolver;
```

31

Adding Objects

Create a `btRigidBody` and add it to the `btDynamicsWorld`. To construct a `btRigidBody` or `btCollisionObject`, you need to provide:

- Mass, positive for dynamic moving objects and zero for static objects
- `CollisionShape`, like a `Box`, `Sphere`, `Cone`, `Convex Hull` or `Triangle Mesh`
- Material properties like friction and restitution

32

Adding a Rigid Body body

```
btRigidBody::btRigidBodyConstructionInfo
 rbInfo(mass, myMotionState, colShape,
  localInertia);
btRigidBody* body = new
 btRigidBody(rbInfo);
dynamicsWorld->addRigidBody(body);
```

33

mass, localInertia and myMotionState

```
btScalar mass(1.f);
btVector3 localInertia(0, 0, 0);
btTransform startTransform;
startTransform.setIdentity();
startTransform.setOrigin(
  btVector3(2, 10, 0));
btDefaultMotionState*
  myMotionState = new
   btDefaultMotionState(
    startTransform);
```

34

colShape

```
btCollisionShape* colShape = new
  btSphereShape(btScalar(1.));
colShape->calculateLocalInertia(
  mass, localInertia);
```

35

Frame Update

- Once a frame, call `stepSimulation` on the dynamics world.
- The `btDiscreteDynamicsWorld` automatically takes into account variable timestep by performing interpolation instead of simulation for small timesteps.
- It uses an internal fixed timestep of 60 Hertz.
- `stepSimulation` will perform collision detection and physics simulation.
- It updates the world transform for active objects by calling the `btMotionState`'s `setWorldTransform`.

36

Rigid Body Dynamics

Rigid body dynamics is implemented on top of the collision detection module. It adds forces, mass, inertia, velocity and constraints.

- `btRigidBody` is the main rigid body object.
- Moving objects have non-zero mass and inertia.
- `btRigidBody` is derived from `btCollisionObject`, so it inherits its world transform, friction and restitution, and adds linear and angular velocity.
- `btTypedConstraint` is the base class for rigid body constraints, including
  - `btHingeConstraint`,
  - `btPoint2PointConstraint`,
  - `btConeTwistConstraint`,
  - `btSliderConstraint`, and
  - `btGeneric6DOFconstraint`.

37

Object Types

There are 3 different types of objects in Bullet:

1. Dynamic (moving) rigid bodies
   - positive mass
   - every simulation frame it will update its world transform
2. Static rigid bodies
   - zero mass
   - cannot move but can collide
3. Kinematic rigid bodies
   - zero mass
   - can be animated by the user, but there will be only one-way interaction: dynamic objects will be pushed away, but there is no influence from them.

38

Center of Mass

- The world transform of a rigid body is in Bullet always equal to its center of mass, and its basis also defines its local frame for inertia.
- The local inertia tensor depends on the shape, and the `btCollisionShape` class provides a method to calculate the local inertia, given a mass.
- This world transform has to be a rigid body transform, which means it should contain no scaling, shear etc.
- If the collision shape is not aligned with the center of mass transform, it can be shifted to match with a `btCompoundShape`, using the child transform to shift the child collision shape.

39

`MotionState`

- `MotionState`s are a way for Bullet to get the world transform of objects being simulated into the rendering part of your game.
- Your game loop will iterate through all the objects before each frame render. For each object, update the position of the render object from the physics body using `MotionState`s.
- Other benefits of `MotionState`s:
  - Computation involved in moving bodies around is only done for bodies that have moved.
  - You don't just have to do render stuff in them. They could be effective for notifying network code that a body has moved and needs to be updated across the network.
  - Interpolation is usually only meaningful in the context of something visible on-screen. Bullet manages body interpolation through `MotionState`s.

40

`MotionState`

`MotionState`s are used in two places in Bullet.

1. When the body is first created. Bullet:
   - Grabs the initial position of the body from the `MotionState` when the body enters the simulation
   - Calls `getWorldTransform` with a reference to the variable it wants you to fill with transform information
   - Calls `getWorldTransform` on kinematic bodies.
2. After the first update, during simulation. Bullet calls the `MotionState` for a body to move it around
   - Bullet calls `setWorldTransform` with the transform of the body, for you to update your object appropriately
   - To implement one, simply inherit `btMotionState` and override `getWorldTransform` and `setWorldTransform`.

41

`btDefaultMotionState`

- Although recommended, it is not necessary to derive your own `MotionState` from `btMotionState` interface.
- Bullet provides a default `MotionState` that you can use for this. Simply construct it with the default transform of your body:

```
btDefaultMotionState* ms = new
  btDefaultMotionState();
```

42

Render Frames and Physics Frames

By default, Bullet physics simulation runs at an internal fixed frame rate of 60 Hertz. Your render loop will have a different or even variable frame rate. To decouple the render frame rate from the simulation frame rate, an automatic interpolation method is built into `stepSimulation`:

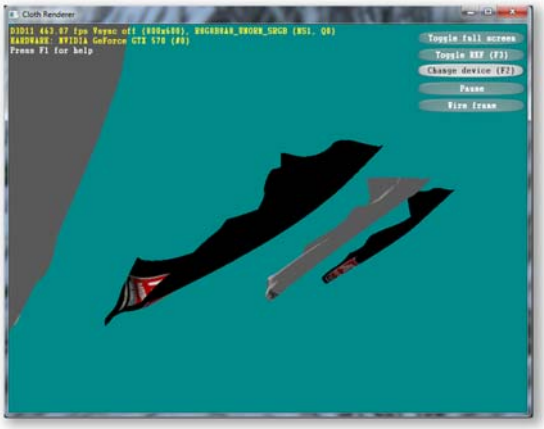
- If the application delta time is smaller then the internal fixed time step, Bullet will interpolate the world transform and send it to the `btMotionState`, without performing physics simulation.
- If the application time step is larger than 60Hz, one or more simulation steps can be performed during each `stepSimulation` call. The user can limit the maximum number of simulation steps by passing a maximum value as second argument.

43

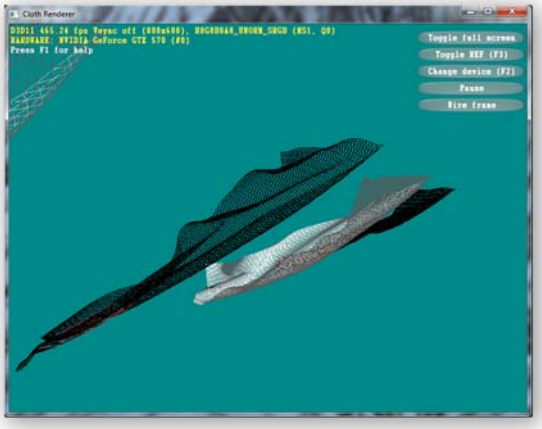
Some Bullet Physics Demos

1. Cloth (DirectX 11)
2. Cloth (OpenCL)
3. Spiders
4. Forklift
5. Ragdoll

44

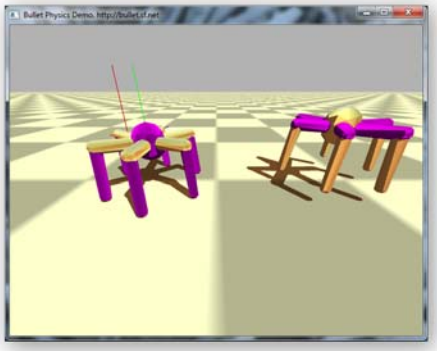


45

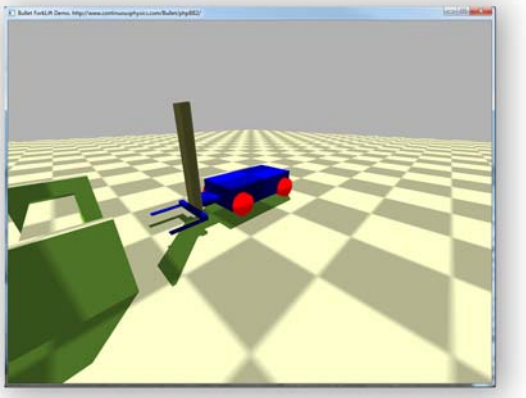


46

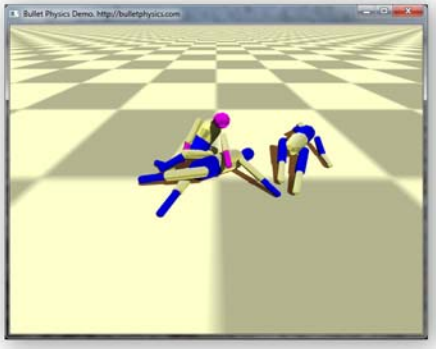
47



48



49



50