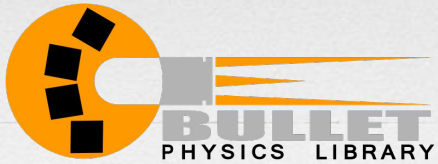
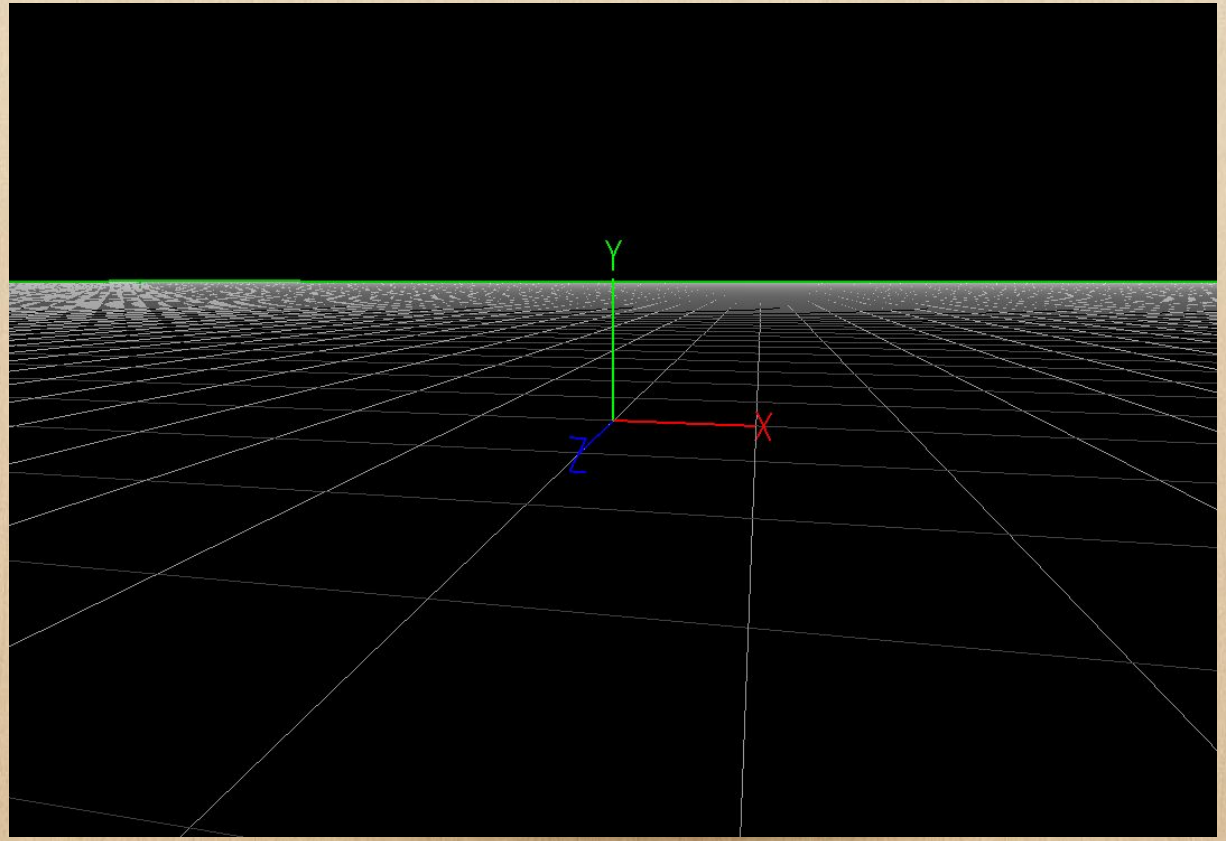

Bullet
integration

Bullet



Bullet is a physics library created by Erwin Coumans. It supports collision detection and soft and rigid body dynamics.

It's used in movies, videogames and other authoring tools. In its github's repository you can find the manuals inside *docs* folder.



Our goal

YOUR TURN !

TODO No 1

```
// TODO 1: Add Bullet common include btBulletDynamicsCommon.h
// ...and the 3 libraries based on how we compile (Debug or Release)
// use the _DEBUG preprocessor define
```

It's exactly the same procedure that we carried out with Box2D. Do you remember? We will use (again) the _DEBUG macro

TODO No 2

```
// TODO 2: Create collision configuration, dispatcher, broad _phase and
// solver
// And destroy them!
```

Bullet pipeline can be completely modulated and customized. We will use the default modules that Bullet offers upon physics world creation.

```
btDiscreteDynamicsWorld(btDispatcher* dispatcher, btBroadphaseInterface* pairCache,
btConstraintSolver* constraintSolver, btCollisionConfiguration* collisionConfiguration);
```

TODO No 2

```
btDiscreteDynamicsWorld(btDispatcher* dispatcher, btBroadphaseInterface* pairCache,
btConstraintSolver* constraintSolver, btCollisionConfiguration* collisionConfiguration);
```

A collision dispatcher iterates over each pair, searches for a matching collision algorithm based on the types of objects involved and executes the collision algorithm computing contact points. Use **btCollisionDispatcher**

The broadphase collision detection provides acceleration structure to quickly reject pairs of objects based on axis aligned bounding box (AABB) overlap. Several different broadphase acceleration structures are available. Use **btDbvtBroadphase**

TODO No 2

```
btDiscreteDynamicsWorld(btDispatcher* dispatcher, btBroadphaseInterface* pairCache,
btConstraintSolver* constraintSolver, btCollisionConfiguration* collisionConfiguration);
```

When different constraints are applied (i.e. when using joints), the velocity and position calculation of each body requires a solver. Use **btSequentialImpulseConstraintSolver** by default.

This module contains default setup for memory, collision setup. For now just use **btDefaultCollisionConfiguration**

TODO No 3

```
// TODO 3: Create the world and set default gravity
// Have gravity defined in a macro!
```

Now, we have all the ingredients to create our world. Create it!

Save the gravity vector as a macro, exactly in the same game that with Box2D. Bullet recommends not to work with objects smaller than 0.2f (1.0f is equal to 1 meter).

With the world created, you can safely uncomment the DebugDrawer.

TODO No 4

```
// TODO 4: step the world
```

Extracted from the manual (page 22):

"By default, Bullet physics simulation runs at an internal fixed framerate of 60 Hertz (0.01666). The game or application might have a different or even variable framerate. To decouple the application framerate from the simulation framerate, an automatic interpolation method is built into stepSimulation: when the application delta time, is smaller then the internal fixed timestep, Bullet will interpolate the world transform, and send the interpolated worldtransform to the btMotionState, without performing physics simulation. If the application timestep is larger then 60 hertz, more than 1 simulation step 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."

TODO Nº 5

```
// TODO 5: Create a big rectangle as ground
// Big rectangle as ground
```

In order to add a new rigidbody, we have to call *world->addRigidBody()*...

...that needs a ***btRigidBody**...

...that is created by **btRigidBody::btRigidBodyConstructionInfo**...

...that accepts three parameters:

- ✗ *Mass* : 1 for default and 0 for static objects
- ✗ *MotionState* : We will use **btDefaultMotionState**
- ✗ **btCollisionShape**: The base class for all shapes in Bullet

TODO No 5

```
// TODO 5: Create a big rectangle as ground
// Big rectangle as ground
```

```
btMotionState *motionState = new btDefaultMotionState();
btBoxShape *shape = new btBoxShape(btVector3(100.0f, 1.0f, 100.0f));

btRigidBody::btRigidBodyConstructionInfo rigidBodyInfo(0.0f, motionState, shape);
btRigidBody *rigidBody = new btRigidBody(rigidBodyInfo);

world->addRigidBody(rigidBody);
```

TODO No 6

```
/ TODO 6: Create a Solid Sphere when pressing 1 on camera position
```

Similar with previous TODO, but now we need to set a position to the body.

To do that, you can add a **btTransform** to the **btDefaultMotionState** constructor…

… that can be obtained from a 4x4 matrix within *glMath* library using the *setFromOpenGLMatrix*( )…

… where you can supply the camera position to this matrix.

Homework

Try to create boxes in the same way as spheres.

And... think on your game! Try to combine these elements: car and physics. Maybe a racing game? Maybe a little Rocket League version? Maybe an obstacle race? Maybe a combination? Bring your ideas the next weekend!

Next Week . . .

Collision Detection