Physics II

CITM

Bullet Physics – Library Integration

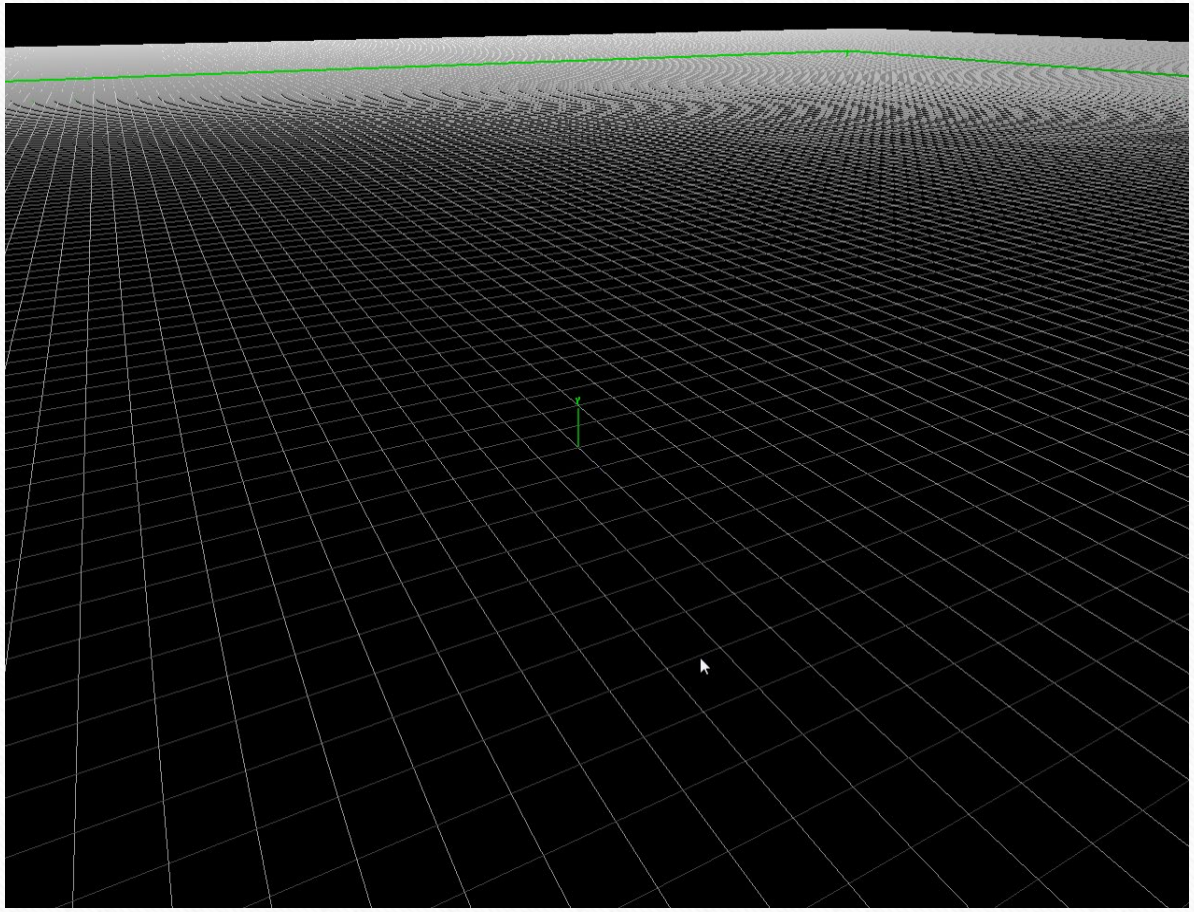
Bullet Physics

- 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 (Wiki).
- In its GitHub's repository you can find the manuals inside docs folder.

- http://www.bulletphysics.org/

What you will have



TODO 1

Add the BulletPhysics libraries and common header.

- Add the 3 library files using: #pragma comment(lib, "...").
- Add the 3 "debug" and "release" versions!
- Include "btBulletDynamicsCommon.h".

TODO 2

Create (+destroy): collision_configuration, dispatcher, broad_phase and solver.

- A “bullet world” can be built with these classes, like “Lego pieces”.
- We’ll need all of them before we can create the world. We can use the default ones provided already by Bullet.

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

TODO 2

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

A **collision dispatcher** detects fine collisions and finds contact points. It also locates the adequate collision algorithm for each pair of objects.

Use **btCollisionDispatcher**

The **broadphase collision detection** does a first pass to detect object that "may collide". Simplifies all objects into spheres or boxes.

There's many ways to do this, we'll use **btDbvtBroadphase**

TODO 2

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

Calculates how the constraints affect the objects attached to them and "solves" how the object and it's restraints interact.

Use **btSequentialImpulseConstraintSolver**

This module contains default setup for bullet, with memory and collision setups.

For now just use **btDefaultCollisionConfiguration**

TODO 3

Create the world (btDiscreteDynamicsWorld) and set gravity.

- With all the pieces ready, let's create our Physics world!
- **#define GRAVITY** as a macro, to access it from anywhere as a constant.
- Set your world's Gravity to your defined value.
- Try to avoid objects smaller than 0.2f (Recommended 1.0f == to 1 m).

TODO 4

Link the “Debug Drawer” functions to the physics world.

- Uncomment all the “Debug Drawer” functions.
- Link them to your physics world.
- It’s not really spectacular yet, though.

TODO 5

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 6

Create a big rectangle as ground.

- To add a rigidbody: World->addRigidBody(**btRigidBody***);
- To create a rigidbody: btRigidBody(**btRigidBodyConstructionInfo**);
- For construction info: btRigidBodyConstructionInfo(**float** mass, **btMotionState***, **btCollisionShape***);

TODO 6

```
btRigidBodyConstructionInfo(float mass, btMotionState*, btCollisionShape*)
```

- **Mass**: Mass of the object. 0 for static objects, 1 by default.
- **btMotionState**: holds position, velocity, inertia… of the body. We can use **btDefaultMotionState**.
- **btCollisionShape**: defines shape of the body. We can use a **btBoxShape**.

TODO 6

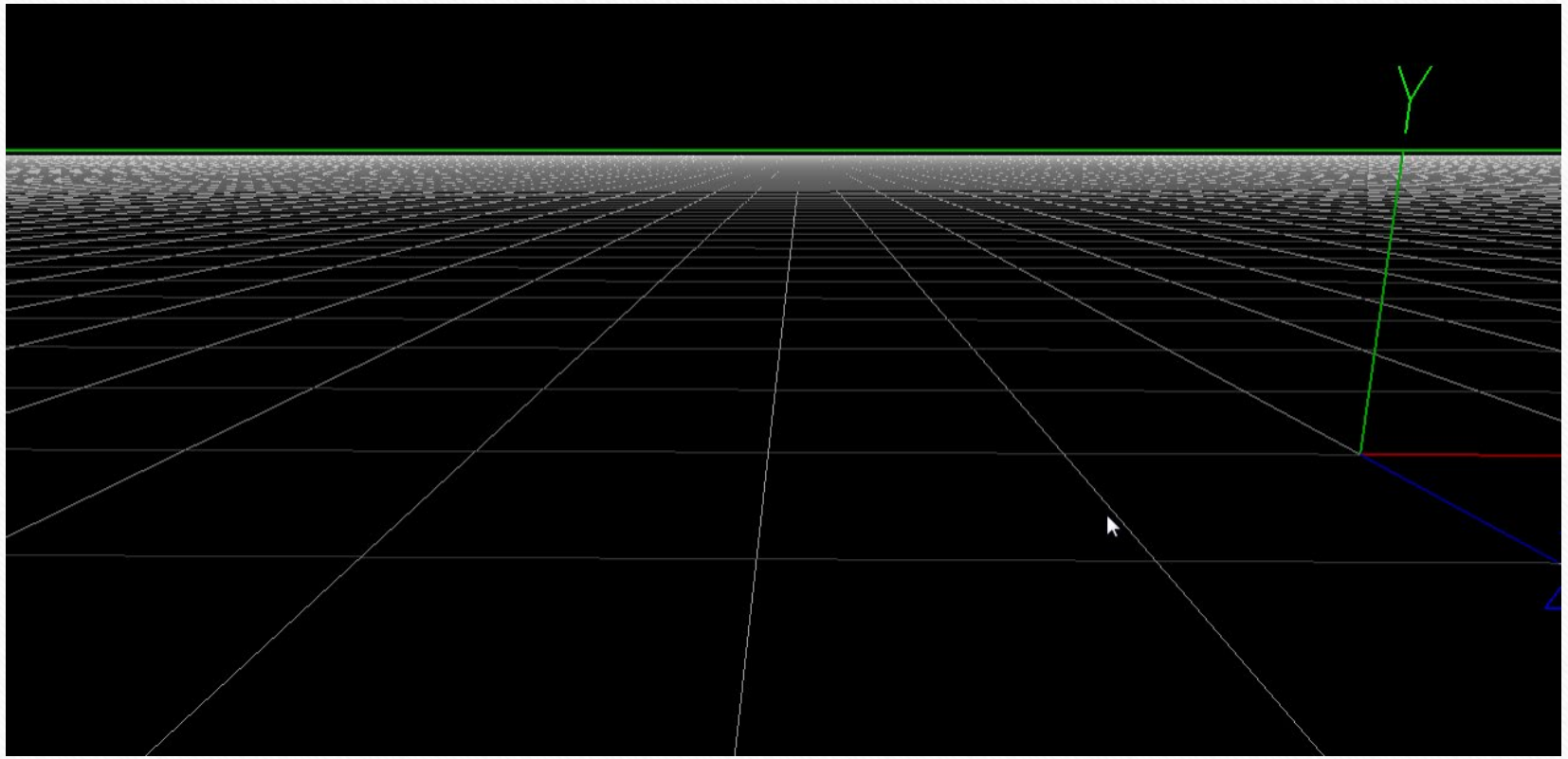
Create a big rectangle as ground.

```
btMotionState *motionState = new btDefaultMotionState();
btBoxShape *shape = new btBoxShape(btVector3(200.0f, 1.0f, 200.0f));

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

world->addRigidBody(rigidBody);
```

TODO 6 - Result



TODO 7

Create a Solid Sphere when pressing 1, on camera position.

- Similar to previous TODO, but:
  - Remember to set mass to “not 0”.
  - We’ll use a **btSphereShape**.
  - We need to set a position to the body.
- We mentioned **btMotionState** was holding the object’s position…

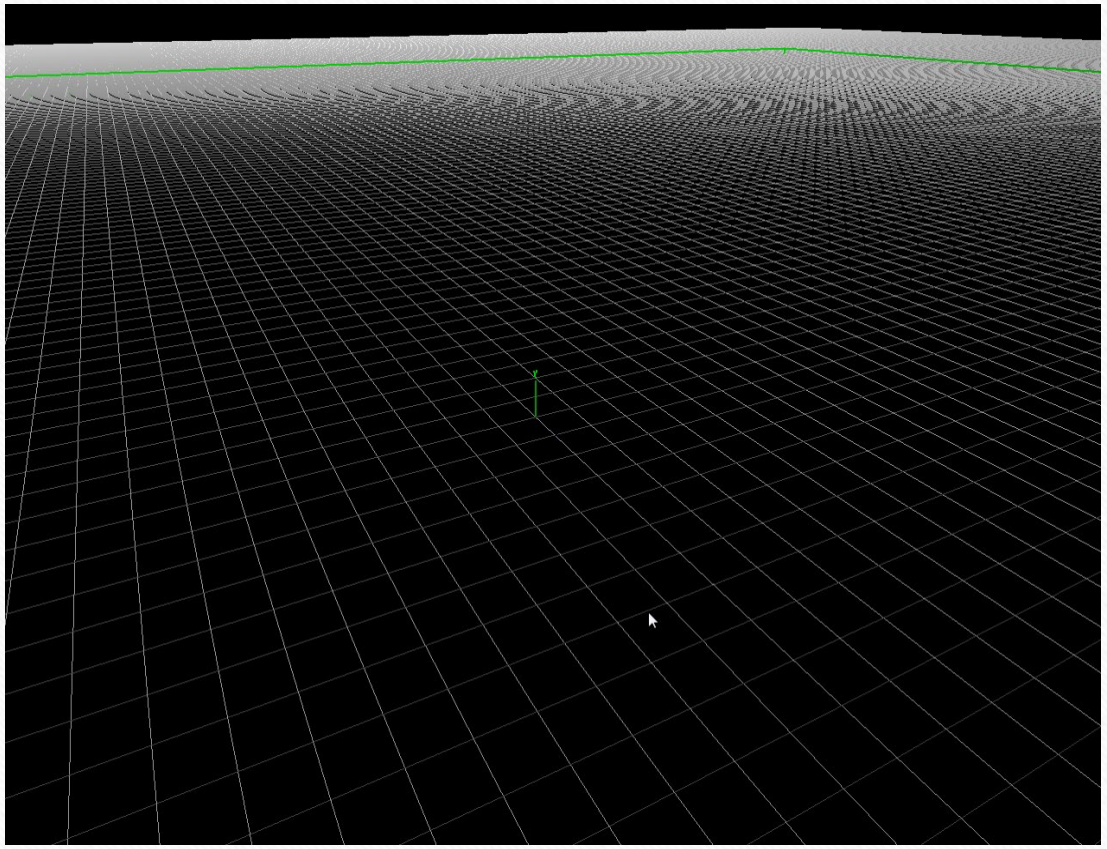
TODO 7

Create a Solid Sphere when pressing 1, on camera position.

- **btMotionState** holds the position, which we want to modify.
- **btMotionState** can be constructed (or set) with a btTransform .
- **btTransform** has the function setFromOpenGLMatrix(...)
- glMath library allows us to create **mat4x4** (4 by 4 matrix, as used by OpenGL)
- **mat4x4** can be initialized to the IdentityMatrix, and translated by the **camera position**.

Where we're at

We can spawn any physics body we want, as longs as a **btCollisionShape** exists for it!



Where we're at

However, right now the code looks something like this:

Homework

- Try to create some boxes. Experiment with different shapes!
- Try adding LocalInertia… it can be calculated from every Collision shape. "calculateLocalInertia"

- And… think on your game! The only requirements are: car and physics.
- Racing game? Mini rocket League? Obstacle course? A trailer carrying stuff?