3D Computer Graphics
IG3D Practical - Rigid Body Simulation

Kiwon Um, Telecom Paris

In this practical, you will learn how to implement a basic rigid body simulator in three-dimensional (3D) space. This practical starts with a provided codebase where basic libraries such as 3D vector and matrix as well as a simple rendering routine are already implemented. The first goal of this practical is to achieve a rigid body animation that is simulated via your implementation following physic laws. The final goal is not limited. Once you finish all the tasks described here, you can extend your codes further as you want. The potential directions for extension are described in the end.

1 Codebase

Your first task is to understand the overall structure of the given codebase. You are guided to read through the `rigid.cpp` and the other files. You can use or even adapt the given math libraries (shown in Code 1 and 2).

Code 1. 3D vector class in Vector3.hpp

```
template<typename T>
class Vector3 {
  // ...
  // member functions and variables
  // ...
};
typedef Vector3<Real> Vec3f;
```

Code 2. 3×3 matrix class in Matrix3x3.hpp

```
template<typename T>
class Matrix3x3 {
  // ...
  // member functions and variables
  // ...
};
typedef Matrix3x3<Real> Mat3f;
```

The main solver is implemented in the `RigidSolver` class. You do not need to implement the entire simulation routine. As shown in Code 3, the given codes perform a very simple update-render routine, which iteratively calls your step function and render function to update the state and redraw the new state using the OpenGL APIs. If you want to have a better viewer, e.g., having a navigation camera, you can modify the rendering routine; but, it is not required.

Code 3. Rigid body solver class in rigid.cpp

```
class RigidSolver {
public:
  // ...
  void step() { /* ... */ }
};

// ...
void render() { /* ... */ }
void update() {
  if(!gPause) solver.step();
  glutPostRedisplay();
}

int main(int argc, char **argv) {
  // ...
  glutDisplayFunc(render);
  glutIdleFunc(update);
  // ...
}
```

Code 4. Build and run

```
  mkdir build # under your main source
      directory where CMakeLists.txt exists
  cd build
  cmake ..
  make
  ./rigid
```

1.1 Build and run

The given codebase uses *cmake* as a build system. You can easily build the executable via general cmake commands. (See Code 4.)

If everything works on your machine, you should be able to see a simple initial simulation setup as on the left of Fig. 1; the middle and right of Fig. 1 are examples of simulation result you may achieve if you implement important functions properly. You can use `q` to quit, `p` to toggle pause of your simulation, `r` to reset your simulation, and `s` to save a screenshot of the current frame into a file.

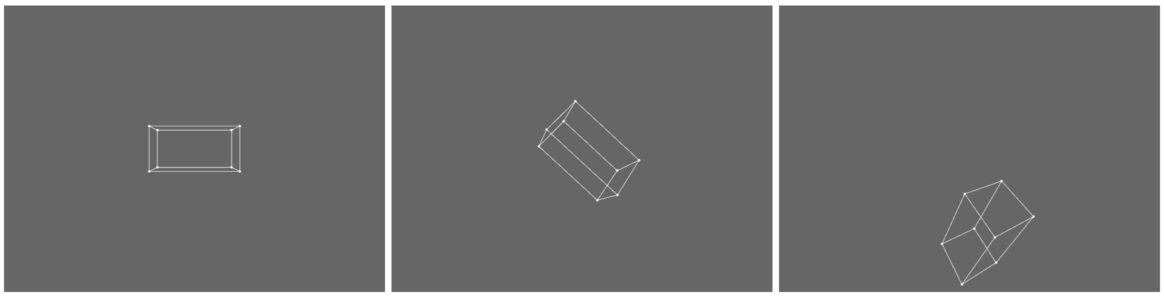

Fig. 1. Screen captures of (left) the first frame and (middle and right) two frames after certain numbers of simulation steps.

2 Rigid body attributes

The very first task is to properly calculate the rigid body's attributes. You can see the member variables of the `Box` class as well as its parent, `BodyAttributes`. The list of important attributes you need to assign is as follows:

- `M`: Mass M
- `I0` and `I0inv`: Inertia tensor and (its inverse) in body space I^o and $(I^o)^{-1}$

You also need to initialize other member variables properly.

3 Time integration

Your solver uses the simulation step of 0.01 and the gravitational acceleration of -9.8m/s^2 by default. See the `RigidSolver` class. You start with the temporal integration of linear momentum (or velocity). See the `RigidSolver::step` function. You do not need to follow the order described below. Make sure that your simulation properly handles mandatory aspects of motion.

4 Force

Forces make the rigid body move. If no force acts on the body, the body will never move unless its initial velocity is nonzero, which is the case of the given codes. You need to implement the force calculation. This will change the momentum thus make the body move. Do not forget the gravitational force always exists. **Mission**: you should implement an initial force of $[50, 150, 0]^\top$ acting on the 0^{th} vertex of your rigid body.

5 Linear momentum

It must be straightforward. You need to handle the linear momentum according to what you have learned from the lectures.

6 Torque

The individual force acting on a vertex of the body should also generate torque thus could produce rotational motion of the body. You need to handle the rotational motion as well. This is the most fun part!

7 Angular velocity

It must be straightforward. You need to handle the angular momentum according to what you have learned from the lectures.

8 Different rigid bodies

The given codebase only handles a cuboid box. You need to extend it to other types of rigid body. A good addition is a cylinder. Referring to the `Box` and `BodyAttributes` classes, implement yourself other type(s) of rigid body (e.g., cylinder and octahedron), and enable the user to choose a simulation target using number keys 1, 2, and so on.

9 Quaternion

You have leaned two different ways to handle the rotational motion. The given codebase uses matrix by default. In this task, you will replace it with quaternion. Implement yourself a quaternion class with mandatory functions, and use the quaternion within the solver. It is recommended you to enable the user to choose one of them before simulation starts.

10 Optional: Generating videos

An animation is nothing complex but a sequence of frames you simulated. You can adapt the given codebase such that it can save a sequence of simulated frames. Once you have collected a set of images, you can use the video editor of *Blender* or any other tools you like, which can convert images into a video file.

11 Extensions

There is no limitation in this practical. You are strongly encouraged to further investigate any extensions you are interested in. You can find a list of potential directions in the following:

- Applying adaptive time step sizes (for real-time applications)
- Putting walls and handling collisions
- Handling multiple objects with their interactions
- ...