Fundamentals of Computer Graphics
Exercise - Rigid Body Simulation

Kiwon Um, Telecom Paris

In this exercise, you will learn how to implement a basic rigid body simulator in three-dimensional (3D) space. This exercise starts with a provided codebase where basic libraries such as a 3D vector and matrix as well as a simple rendering routine are already implemented. The goal of this exercise is to achieve a simple simulator that animates a rigid object based on the physics laws. Once you finish all the tasks described here, you can extend your codes further as you want. The potential directions are described in the end.

1 Codebase

Your first task is to understand the overall structure of the given codebase. Please read through the `main.cpp` file and the others. You can use or even adapt the given math libraries: `Vector3.hpp` and `Matrix3x3.hpp`.

The main solver (`RigidSolver`) is implemented in the `RigidSolver.hpp` file, and the simulation routine is implemented in the `main.cpp` file. As shown in Code 1, the given codes perform an update-and-render routine, which iteratively calls the solver's `step` and the global `render` functions to update the state (such as position and orientation of the rigid body) and redraw the new state using the OpenGL APIs.

Code 1. Rigid body solver routine

```
// ... RigidSolver.hpp ...
class RigidSolver {
public:
  // ...
  void step(const tReal dt) { /* ... */ }
};

// ... main.cpp ...
void render() { ... }
void update(const float currentTime)
{
  // ...
  g_scene.solver.step(std::min(dt, 0.017f));
  g_scene.rigidMat = g_scene.rigidAtt->worldMat();
}
// ...
```

Code 2. Build and run

```
cmake -B build # or mkdir build; cd build; cmake ..
make -C build  # or make
./tpRigid
```

1.1 Build and run

Important! This guideline is written for Linux systems. If you use other operating systems, you should adapt it accordingly.

The given codebase uses *cmake* as its build system. You can easily build an executable via general cmake commands. (See Code 2.)

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 example screenshots of a simulation result you may achieve if you finish important tasks properly. You can use Esc to quit, P to toggle pause of your simulation, R to reset/restart your simulation, and S to save a screenshot of the current frame into a file. Try H to see the help.

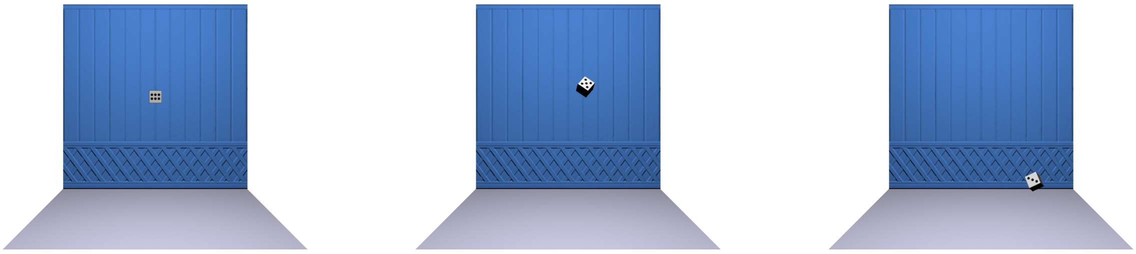

Fig. 1. Screen captures of (left) the first frame and (middle and right) two frames after a certain number of simulation steps.

2 Rigid body attributes

To begin with, you need to properly calculate the rigid body's attributes. You can see the member variables of the `Box` class as well as its parent, `BodyAttributes`, in `RigidSolver.hpp`. The important attributes you need to assign are 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 an arbitrary gravitational acceleration of -0.98 by default. See the `Scene` and `RigidSolver` classes. You start with the temporal integration of linear momentum (or velocity). See the `RigidSolver::step()` function. You do not need to follow the order of tasks 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. You need to implement the force calculation. This will change the momentum hence will make the body move. You can first take into account the gravitational force. Then, implement an arbitrary initial force:

- You implement an instant force of $[0.15, 0.25, 0.03]^\intercal$ acting on the 0^{th} vertex of your rigid body at the 1^{st} step.
- You can play with the instant force changing its strength and direction.

5 Linear momentum

It must be straightforward. You need to handle the linear momentum according to what you have learned from the class. If everything is correct, you should be able to see a smooth parabolic motion of your object if you apply the initial force specified above.

6 Torque

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

7 Angular velocity

It must be straightforward. You need to handle the angular momentum too according to what you have learned from the lectures. If everything is correct, you should be able to see that your object is rotating while traveling along the smooth parabolic trajectory.

8 Quaternion

Once you take into account the rotational motion, you may observe that your object is quickly distorted over time even after a few steps. As discussed in the class, this is due to the numerical error that deteriorates your rotation matrix. To address this problem, you have learned two different ways. In this task, you should employ the quaternion. Implement a quaternion class with important functions such as multiplication, normalization, and conversion (to a rotation matrix). You should use your quaternion within the solver.

9 Extensions

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

- Handling other types of rigid bodies such as a cylinder and an octahedron
- Handling collisions with walls
- Handling multiple objects with collisions
- ...

10 Submission guideline

Once you finalize your exercise, you need to submit a packed/compressed file via eCampus:

- by the midnight (23:59) on Tuesday 2 April 2025

Please make sure that your implementation compiles without errors and the application runs as you programmed. Your package must contain:

(1) Your final implementation files for the solver, `RigidSolver.hpp`,
(2) Any additional files only if you added on purpose
(3) A short PDF (maximum 2 pages) report (written in English) that contains a summary of what you achieved and screenshots you took from each task.
(4) A short video file (maximum 1 minute) that records an animation of your final implementation.

IMPORTANT: DO NOT include unnecessary files such as the executable and object files generated from your build.