CS171 Assignment 5: Rigid Body Simulation

Introduction

Rigid body simulation is responsible for simulating the dynamic motion (basically the translation and rotation) of one or more rigid objects moving over the space with contact constraints enforced among them. In this assignment, you are required to implement a very basic rigid body simulation involving translation of spheres with collision detection and contact handling between spheres and parallelograms.

In the following, we will give you the specifics about what you need to accomplish, as well as some related guidelines in order to assist your programming.

Programming Requirements

- **[must]** Synchronize the simulation and the rendering in OpenGL to show the result in real time.
- **[must]** Implement the collision detection between the parallelograms and the sphere.
- **[must]** Implement the collision adjustment so that the inter-penetration is within the given tolerance (a small threshold).
- **[must]** Implement the colliding contact handling between the the parallelogram and the sphere without consideration of rotation.
- **[optional]** Simulate with multiple spheres simultaneously.
- **[optional]** Simulate a cube in the scene with rotation taken into account.

Notes

- We will offer a code skeleton for you. Please make sure you understand the functionality of each declared class and method in the code skeleton before you start. You can write your own code without the skeleton, but you must at least simulate the same scene we give you, which is important for us to verify the correctness of your implementation.
- To verify your implemented algorithms, you can use various scene settings.
- Please note that you are **NOT** allowed to use third-party libraries except for the libraries we give in the skeleton code.

Submission

You are required to submit the following things through the GitHub repository:

- Project scripts in the Coding folder.
- A PDF-formatted report which describes what you have done in the Report folder.

Submission deadline: **22:00, Dec 25, 2021**

Grading Rules

- You can choose to do the **[optional]** item, and if you choose to do it, you will get additional scores based on the additional work you have done. But the maximum additional score will not exceed 20% of the entire score of this assignment.

- **NO CHEATING!** If found, your score for the entire assignment is zero. You are required to work **INDEPENDENTLY**. We fully understand that implementations could be similar somewhere, but they cannot be identical. To avoid being evaluated inappropriately, please show your understanding of code to TAs.
- Late submission of your assignment will be subject to score deduction.

Skeleton Project/ Report Template

- The skeleton program and report template will be provided once you accept the assignment link of GitHub classroom which we published on the Piazza. If you accept the assignment through the link properly, a repository which contains the skeleton project and report template will be created under your GitHub account.
- Please **follow the template** to prepare your report.

Implementation Guide

Here we will mainly cover some key functionalities you are required to complete, but we suggest you read the entire code skeleton in case your implementation is not compatible with the skeleton. In addition, this guide will not cover the optional parts, which mostly count on your individual study and understanding.

Git Classroom

Accept the assignment in this link through Git classroom to start your assignment or you can download the zip package.

1. Make your scene move

The first step in this assignment is to make the scene move without considering any collisions. You can notice that in each time step, the function `Scene::update` is called. You can first try implementing this function by updating each rigid body in the scene state forward with the function `Sphere::Forward` (Since the wall is static, `Wall::Forward` is empty).

To show the simulation result in real time, you should update the `RigidBody::opengl_object` immediately when you update the simulation state.

When you successfully implemented `Scene::update` together `Sphere::Forward`, you can find you scene is updated in the created OpenGL window once you press the space key down.

Related files: **scene.cpp**, **rigid_body.cpp**

2. Detect the collision

The second step is to detect the collision in the scene. You are supposed to implement the function `CollisionHandler::CheckParlgrmAndSphere`.

Related files: **collision.cpp**

3. Adjust the collision point

Most of the time, the detected collision is not perfect. In other words, there may be inter-penetration between the two collided rigid bodies due to discrete time step. We define a tolerance (a small threshold value) to restrict the degree of inter-penetration. You should implement the collision adjustment in the function `Scene::AdjustCollision` to ensure the degree

of inter-penetration is within the tolerance. You may need to implement `Scene::Backward` and `Sphere::Backward` to facilitate the implementation of `Scene::AdjustCollision`.

You can use numerical method of *bisection* to adjust the collision. For more details, please refer to Section 6 in this note.

Related files: **scene.cpp**, **rigid_body.cpp**

4. Handle colliding contact

After the collision is detected and adjusted, the next step is to handle the collision contact. You are required to implement the function `CollisionHandler::Handle`.

You can refer to Section 8 in this note for handling colliding contact.

Related files: **scene.cpp**, **collision.cpp**

Expected Results

We provide here five scene settings, where you can successfully simulate the scenes with ids 0, 1, 2 without considering the interactions between spheres, and you need to finish the first optional part to simulate scenes with ids 3 and 4.

The expected results are shown below.

Scene id = 0

0:00 / 0:06

Scene id = 1

0:00 / 0:10

Scene id = 2

0:00 / 0:38

Scene id = 3

0:00 / 0:21

Scene id = 4

0:00 / 0:09