GAMES103 Lab 1: Angry Bunny

Due Date: 12/06/2021, 11:59PM

In this lab assignment, you will learn how to write a rigid body dynamics solver, WITHOUT using the built-in physics engine in Unity. Please go through the slides and make sure you can grasp all of the details before you continue.

Once you download the example package and import it into an empty Unity project, you should see the scene file and the bunny mesh model already being loaded. In the script file, you should also see some functions and variables being defined already for your convenience.

You can use basic arithmetic features provided by Unity, but keep in mind that Unity has some limitations, e.g., no quaternion addition/subtraction, no 3-by-3 matrices, etc.

Your goal is to implement rigid body dynamics in the script. The game works as follows: the bunny initially does rotational motion only at its starting position $[0, 0.6, 0]$; after the player presses 'l', the bunny will be launched and start to perform both translational and rotational motion; when the player presses 'r', the bunny will return to its starting position, waiting for the next launch.

1 Basic Tasks

1.a. Position update (2 Points) In the `Update` function, implement the update of the position and the orientation by Leapfrog integration. Disable the linear motion and the position update, if *launched* is false.

(Hint: In general, try not to directly modify variables in `transform`, as doing this can occasionally slow down your simulation. Use temporary variables instead.)

1.b. Velocity update (2 Points) Calculate the gravity force and use it to update the velocity. To produce damping effects, you can multiply the velocities by linear and angular decay factors: $\mathbf{v} = c^{\texttt{linear\_decay}}\mathbf{v}$ and $\boldsymbol{\omega} = c^{\texttt{angular\_decay}}\boldsymbol{\omega}$. Given the gravitation being the only force, you don't need to calculate torque or to update the angular velocity. Now after you launch the bunny, you should see the bunny flying and spinning (with some initial angular velocity)!

1.c. Collision detection (2 Points) In your `Collision_Impulse` function, calculate the position and the velocity of every mesh vertex. Use them to determine whether the vertex is in collision with the floor. If so, add it to the sum and then compute the average of all colliding vertices.

1.d. Collision response (3 Points) In the same function, apply the impulse-based method to calculate the proper impulse $\mathbf{j}$ for the average colliding position. You then update the linear velocity and the angular velocity by $\mathbf{j}$ accordingly. Doing this will enable the collision with the floor.

1.c. Collision with the wall (1 Point) Can you enable the collision with the side wall as well?

2 Bonus Tasks (3 points)

In another script file, you should see an incomplete dynamics solver based on shape matching. This solver is essentially a particle system, in which every particle has its own position and moves with its own velocity. The secret is in the polar decomposition function (already provided in the package)

for you to extract rotation out of a linear transformation estimation. Based on the slides and the paper, can you implement this idea?

(Hint: Shape matching has difficulty in precisely controlling frictional contact outcomes, so don't worry if you see minor sliding effects.)

3 Submission Guideline

Save all of your files, including scene and script files, and export them into a package. Name your package as lastname_firstname_Lab1.unitypackage and email it to games103@style3D.com.