Daniel Hance – Physics A02 Report

Overview

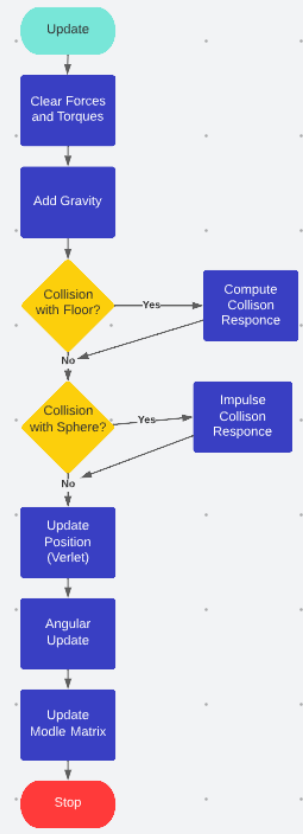
The physics engine that I have created can simulate multiple rigid body spheres with a large number of customizable parameters which can be edited in .txt file in the projects folder and the engine can calculate both linear and angular rigid body motions. All spheres can handle collisions between themselves and other spheres using collision impulse responses and collisions between themselves and the floor using a sphere to plane collision responses. Gravity, friction and velocity gained from a collision are the three forces that act on the spheres with friction also adding torques for angular momentum.

Research

Before starting to program the physics engine I needed to work out what it needed to accomplish. I decided on a program that can handle rigid body collision responses between spheres and planes/spheres and simulate forces such as gravity, friction, velocity and angular velocity. I then did some research to find algorithms that I would need for both physics calculations and other areas of the engine. First I did research into finding out how to get variables using values from a .txt file which I found a suitable algorithm for online[1]. I got a large amount of the physics simulation algorithms from physics lectures on brightspace[2], these included Euler, Runge Kutta 2, Runge Kutta 4, Verlet, and algorithms for sphere to sphere and sphere to plane collision detection and responses.

High Level Design Description

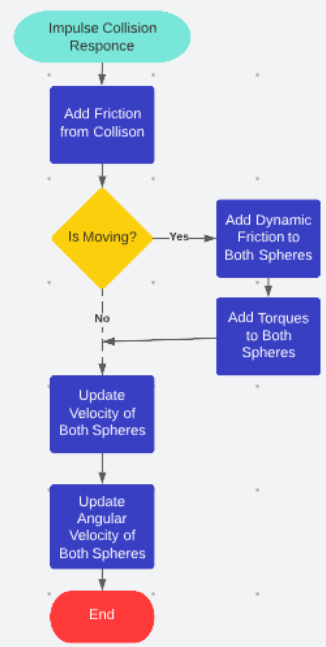
Although the physics engine is made up of multiple files the two most important ones are scene.cpp and dynamicObject.cpp, the scene script handles creating all of the physics objects, reading data form the .txt files and calling each of the physic objects update function each frame in runtime. As this script does not handle any of the actual physics calculations this description will focus on dynmaicObject.cpp and its functions.



The physic object starts its simulation when its update function is called, this function calls all the other functions within the object. First all forces and torques acting on the object are cleared so a fresh simulation can be started and then gravitational force is added. Next the ComputeCollisonResponce() function is called which will first check for a collision between the object and a plane at y-level 0 (the floor), if a collision is detected a contact force is added to the object along with a friction force, then torques are calculated and added using the friction force.

A sphere-to-sphere collision is then checked, if this is true ApplyImpluseResponce() is called to calculate the collision response from this collision. In ApplyImpluseResponce() frictional forces and torques are added to both objects if they are moving. Both objects then have their

velocity and angular velocity updated, this calculates all relevant forces needed to handle the collision response, this can be seen in the impulse collision response flowchart.



Having completed all collision detection the Update() function now calls Verlet() to update the object's position and velocity using all the forces added in previous steps, while the Verlet algorithm is currently being used in the engine Euler, Runge Kutta 2 and Runge Kutta 4 have been implemented so this can be replaced with any of them. The Verlet algorithm has been used over the other as I found through testing that it produced the most realistic position updates in a wide range of situations. AngularMotionUpdate() is now called to update the objects rotation using all the torques added in the previous steps.

Finally UpdateModelMatrix() is called, this function has no physics calculations in it and just updates the physic objects model using all the physic values calculated thought the update, once this is completed the Update() function is finished and scene.cpp can run the script to render the object to the screen.

Evaluation

Overall, the physics engine that I created is effective in simulating multiple aspects of rigid body sphere physics including, linear and angular motions, collision impulse responses between both planes and spheres and using .txt file to take inputs to allow for a wide range of situations to be simulated. Despite this the physics engine is still missing some features that would allow for more realistic simulations, mainly, not including other rigid body shapes (cubes), and collision response between the spheres and angular planes are still missing. Going forwards the features I plan on adding in future development are .txt file input validation, angular plane collision response and graphically improving the simulation area.

References

[1] - *C++: Read from text file and separate into variable* (1957) *Stack Overflow*. Available at: https://stackoverflow.com/questions/3946558/c-read-from-text-file-and-separate-into-variable (Accessed: April 26, 2023).

[2] - Tang , W. (no date) *Teaching & Learning Materials*, *Brightspace*. Available at: https://brightspace.bournemouth.ac.uk/d2l/le/content/298828/Home (Accessed: April 14, 2023).