Edinburgh Napier UNIVERSITY

Coursework Report

Arran Smedley
40406581@napier.ac.uk
Edinburgh Napier University - Physics Based Animation (SET09119)

Abstract

The purpose of this project is to learn, understand and create realistic collisions with sphere objects within OpenGL C++. By the use of the lecture's tutorials and practicals taught throughout the semester i will apply this knowledge to create a realistic pool simulation (sphere collisions).

Keywords – OpenGL, Pool, C++, Coursework, Project, Bounding Volumes, Bounding Volume Hierarchies, Quad Trees

Figure 1: **Inspiration** - Inspiration used for my scene.

1 Introduction

Primary Algorithms The primary algorithms used within this project consisted of, conservation of momentum, Bounding Spheres and Bounding Volume Hierarchy learned through the rigid bodies and collision topics within the module.

2 Related Work

For the most part of this coursework i made use of the past topics in which we had learned, particles, mass spring systems, rigid bodies and collisions in which i used the framework in which i created different classes such as the rigid body class in order to aid with this coursework.

3 Implementation

In order to make this scene a good variety of physics learned from the entire module were implemented to make this scene look how it is, and these are the tasks in which i completed within the coursework :

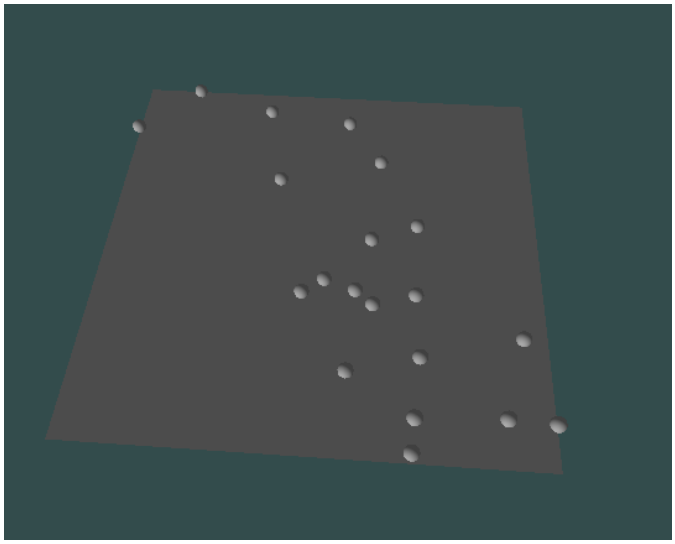

Figure 2: **Actual Scene** - The main scene of my project

3.1 Sphere Collision

For this part of the coursework i made use of the past rigid bodies coursework in which i created a rigid body colliding against a plane. The difference with this coursework is that it was multiple spheres colliding with other spheres in an enclosed area (cushions to keep the balls within a 30x30 area). The way in which i started this project was by creating an array of rigid bodies (balls). I done this by simply using prior knowledge of creating multiple particles within the mass spring systems, by creating an array variable of type rigid body and making it so that it holds 20 balls. Next i implemented a for loop that set the shader, mesh, mass, radius, velocity and position for each ball, ensuring that the position and velocity were randomised as specified in the brief with the position being between 0 and 30 on the x and z axis (on the table) and the velocity between 0 and 20. To ensure no balls were overlapping i came up with an algorithm in which makes it so that for every ball on the table if they are overlapping /colliding (Position is relatively the same) then randomise there location again.

Now onto the movement of each ball. As shown through prior projects within this module, i have used the somewhat same structure through most of them. This project was very similar as i involved an integrate function in which calculates the velocity and forces applied to the rigid-body.

For the collision algorithm i used Bounding Sphere's through

the knowledge of the lecture slides provided. As they are simple, memory efficient, provide inexpensive intersection tests and rationally invariant. I done this shown in the C++ code bellow.

Listing 1: Bounding Sphere's

```
#include <iostream>

for (int i = 0; i < maxballs; i++)
    {

        if (rb[i].getPos().x >= 30|| rb[i].getPos().x <= -30 || rb[i].getPos().z >= 30 || rb[i].getPos().z <= -30)
        {
            rb[i].setVel(-rb[i].getVel());

        }
        for (int j = 0; j < maxballs; j++)
        {
            if (i != j)
            {
                if (distance(rb[i].getPos(), rb[j].getPos()) <= 2.5)
                {
                    rb[i].setVel(-rb[i].getVel());
                    rb[j].setVel(rb[j].getVel());
                }
            }
        }

    }
```

To explain the logic behind this code is that for every ball if they are outside the boundary of 30 then set there velocity to the opposite velocity of what they had (so they go in the opposite direction). Calculate the distance of each ball to each ball and ensure that its more than the radius of both balls to then apply a collision(so that it makes a realistic collision and no balls overlap each other), set the velocity of ball 1 to the opposite and ball 2 to the opposite to ensure both balls go different direction after collision has happened.

3.2 Scalability

For this part of the coursework i have to ensure that the approach/method i use deems useful when it comes to optimisation, to be able to handle a plane of up to 1000x1000 and up to 100 balls without lag.

At an attempt at this i did not get a working program. Though i did somewhat understand the logic behind it. What i attempted was a quad tree algorithm. What a quad tree is, is a spatial indexing technique. In a Quadtree, each node represents a bounding box covering some part of the space being indexed, with the root node covering the entire area. Each node is either a leaf node - in which case it contains one or more indexed points, and no children, or it is an internal node, in which case it has exactly four children, one for each quadrant obtained by dividing the area covered in half along both axes - hence the name. Inserting data into the quad tree is by, starting at the root node, decide which quadrant your point occupes. Move to that node and repeat, until you find a leaf node. Then, add your point to that node's list of points. If the list goes passed a maximum number of elements, split the node, move the points into the correct sub nodes.

Reasoning behind not being able to produce this was i found it very challenging to translate this knowledge into code and by shown in my source code i have attempted to implement it (just obviously not correctly).

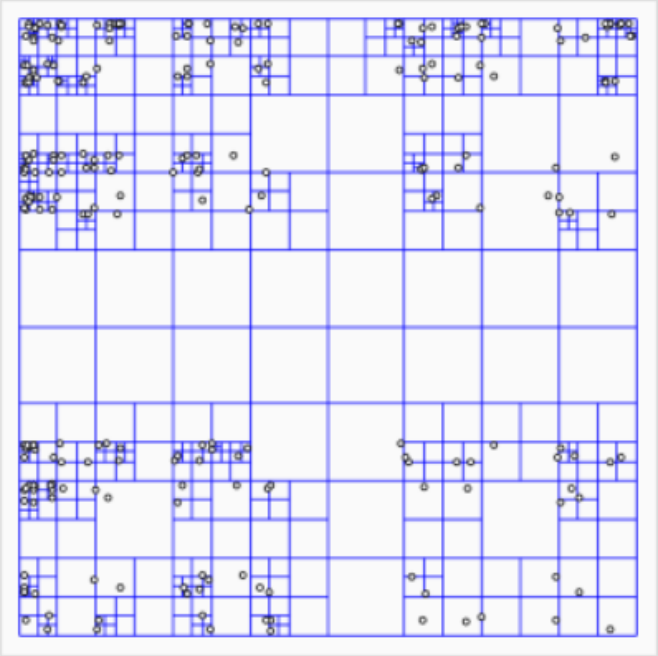

Figure 3: **Visual representation of a quad tree** - Source: wikipedia

4 Future Work

My initial plan was to ensure i got part 1 and 2 fully completed and to be able to somewhat do part 3 if i had time. Though this coursework apposed alot more challenging then i expected, especially the scalability part. I spent too much time researching different ways in which it could be scaled to the point where i was unsure where to start with it. With more time i would've understood Quad trees in a little more depth to be able to implement it into code and then i would've added features such as friction and rotation for part 3.

5 Conclusion

For the majority of part 1 i feel i met the requirements asked of the coursework applying knowledge from lecture's and practicals in order to create a simulation of sphere to sphere and sphere to cushion simulation. For part 2 i did not meet the requirements as i didnt have a functioning simulation to be able to demonstrate. To conclude this project i believe personally i need to critically evaluate how much time i spend researching compared to applying the knowledge to code as i didn't leave enough time to apply the knowledge i learned to code.

References

Quad Trees : https://en.wikipedia.org/wiki/Quadtree
Bounding Volumes : 06 - Collision detection - Part 2 - Bounding Volumes - Author Gregory Lepaltre