2023

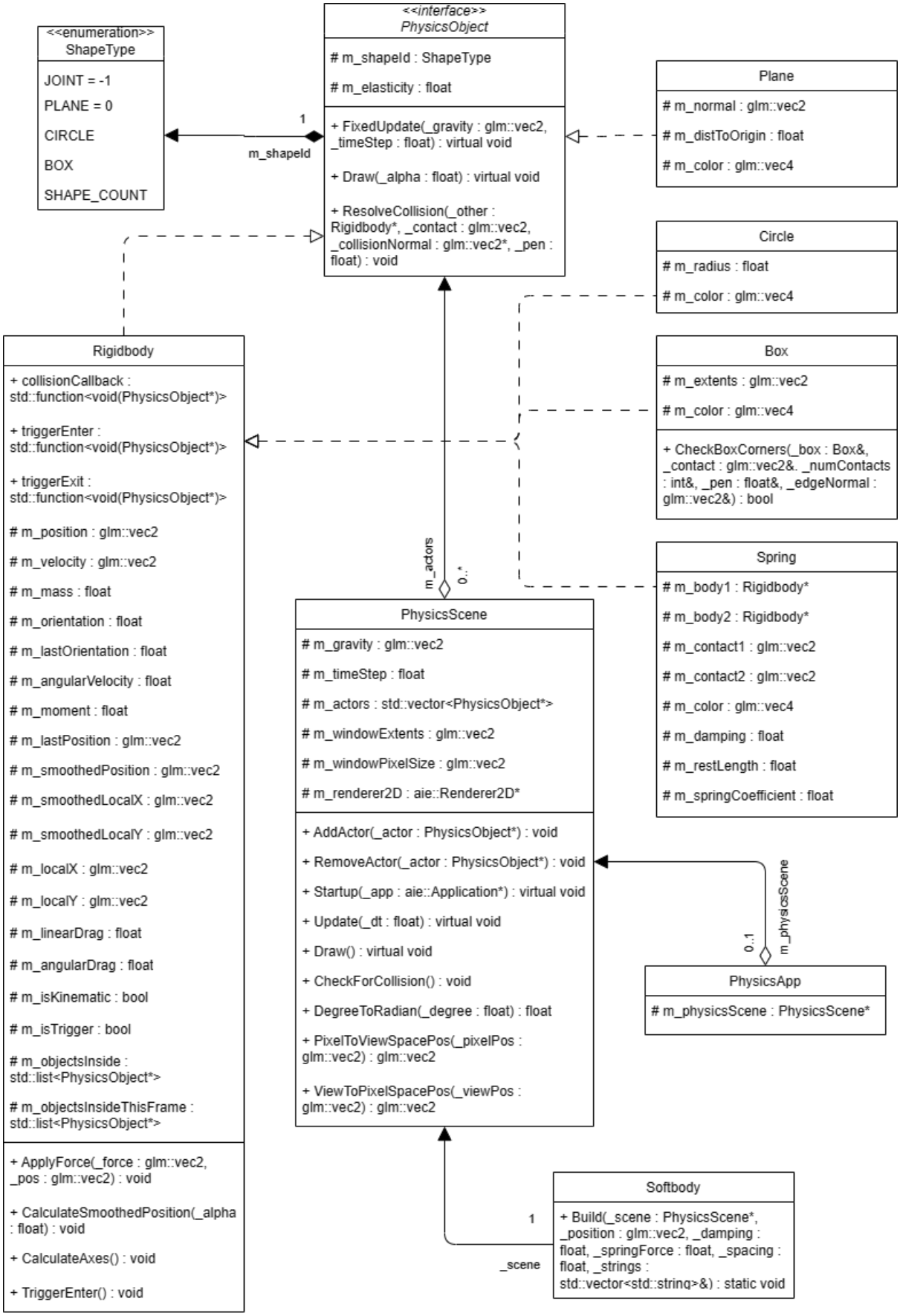
Custom Physics Documentation

8 BALL POOL

ETHAN DAWKINS

Contents

1.0 - Custom Physics Simulation Class Diagram



2.0 - Custom Physics Simulation Interactions

The goal of this physics simulation is to attempt to mirror real-life physics of dynamic and static objects in a few simple example scenarios. This physics simulation is demonstrating how dynamic objects with elasticity, drag and contact forces interact with other dynamic objects in various conditions. It also demonstrates how they interact with static objects; these objects are considered to have infinite mass and cannot be influenced by any external forces.

The dynamic objects involved in this simulation include circles, boxes, springs, and soft bodies. The static objects involved in this simulation include planes and the other primitive shapes when they are set to be kinematic. These objects can interact with each other through the implementation of rigid bodies that abide by Newton's Three Laws. Each rigid body has its own velocity that it uses to move its position, drag is then applied to slow down the rigid body in a realistic fashion. When a rigid body is colliding with another rigid body multiple steps are involved; first a resulting force is calculated using elasticity for each object and applied, then contact forces are applied to separate any objects that are slightly overlapping.

Each physics scene is responsible for simulating all the physics objects within that scene, and appropriately detecting collisions between them. This ensures that the simulation is being handled in a separate section of the app, to allow for easier debugging and more control over how the physics simulation works / behaves. The physics app then has a reference to the current physics scene, to allow handling of multiple physics scenes.

Each shape class is responsible for its individual variables and functions, i.e. the box storing its extents and having a function for checking its corners. Having each shape as its own class also allows for other classes to inherit from the primitive shape types, meaning the base functionality is kept the same across multiple classes of the same shape type. The only exception to this is the soft body class which is not a child class, as it generates an array of springs that make up the shapes form and adds the springs to the passed in physics scene.

All these systems gel and work together to simulate physics in a realistic fashion, while also maintaining structure throughout the libraries core functionalities. Each class is modular allowing for control over exactly how the system works, and also allowing for any modifications to the system where necessary.

3.0 - Custom Physics Simulation Potential Improvements

3.1 - Improvement #1

The first improvement that could be made to the physics simulation is to improve the performance of the collision detection by using spatial hash grids. A hash grid is a way of storing objects so that they can be efficiently retrieved for any collision calculations.

A hash grid simply divides the area into a 2D grid made up of occupied cells, each cell is then responsible for keeping track of which physics objects are currently contained within the cell. Then in the update loop each physics object within each cell is checked against all the other objects within the same cell, to see if any of them are colliding and run any appropriate collision response calculations.

3.2 - Improvement #2

Another improvement that could be made to the physics simulation is adding support for other primitive shapes, i.e. triangles, convex polygons, etc. This would be accomplished by using each edge normal of a shape to find if a point is intersecting with the shape.

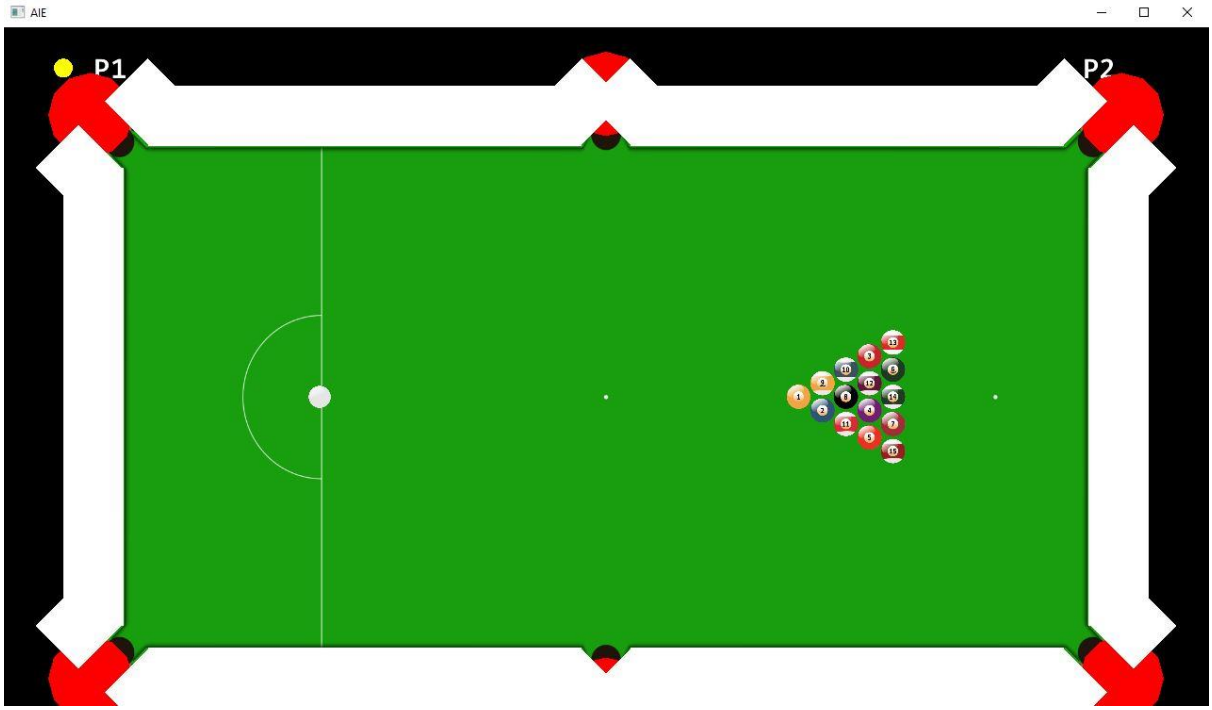
Triangle collision would be a simple case of checking each side using its normal and vertices positions to find if a point is on the inside of the triangle. Performance of triangle collision detection could be further improved by only allowing equilateral triangles to be a part of the physics simulation.

An advantage of convex polygons is that there are no inward pointing faces / vertices, meaning it is a closed shape and can be treated the same as a triangle. Collision detection of a convex polygon is very efficient compared to non-convex polygons, as a point only needs to be on the inside of the shape for collision detection rather than doing expensive calculations to check if it is inside the vertices.

4.0 - Visualised Game Using Your Custom Physics Simulation

The chosen visualisation game is a recreation of 8-ball pool, played by the classic 8-ball pool rules. These rules include pocketing the white ball / hitting the black ball first / hitting the other player's ball first / hitting no balls; all of these situations gives the other player two turns. Potting the black ball before potting all your balls / potting the white after the black; both mean you lose the game.

The pool table is made up of boxes that are set to kinematic for the pool table edges, white in the image below, the pockets are then made up of circles that are set to triggers, red in the image below.



The table and billiards, including the cue ball, are made up of a custom 'Sprite' class that wraps up a texture into an easy-to-use class for rendering textures to the screen. The positions of the billiards and cue ball are hard-coded so they are in the correct position to line up with the background.

To handle the 8-ball pool rules there are separate logic paths for when the cue ball collides with any other billiard, and when a billiard enters a pocket. There are also a few failsafe logic paths, i.e. if a billiard clips through the table edges.

The players' turn is indicated by a yellow circle next to the P1 / P2 text, the counters are made up of the pocketed billiards in the order they are pocketed and placed to the corresponding team counter.

5.0 - Third Party Libraries

The third-party libraries used in the physics application were the OpenGL library for graphics rendering and the GLM library for mathematics used in the application.

Open Graphics Library (OpenGL) is cross-language and cross-platform application programming interface (API) for rendering 2D and 3D graphics.

Graphics Library Mathematics (GLM) is a mathematics library built for use with OpenGL applications, this library provides implementations for matrix transformations, quaternions, vector mathematics, etc.

6.0 - References

Group, K. (no date) *The industry's foundation for High Performance Graphics*, *OpenGL.org*. Available at: https://www.opengl.org/ (Accessed: February 18, 2023).

OpenGL Software Development Kit (no date) *The Industry's Foundation for High Performance Graphics*. Available at: https://www.opengl.org/sdk/libs/GLM/ (Accessed: February 18, 2023).

HashGrid (no date) *giCentre*. Available at: https://www.gicentre.net/utils/hashgrid#:~:text=A%20hash%20grid%20provides%20a%20way%20of%20storing,large%20set%20of%20objects%20occupying%20a%20limited%20space. (Accessed: February 18, 2023).

Singhal, A. (2018) *Making a 2D physics engine: Shapes, Worlds and integration*, *CodeProject*. CodeProject. Available at: https://www.codeproject.com/Articles/1214829/Making-a-D-Physics-Engine-Shapes-Worlds-and-Integr (Accessed: February 18, 2023).