Physics Simulation

This is a simulation that demonstrates static and dynamic rigid body physics programmed using C++ on Visual Studio.

Research done to make the physics simulation included following tutorials supplied by AIE.

Physics Simulation

Fixed Update Loop

* Fixed Update is a function that gets called on a RigidBody.
* It first takes in a vec2 for gravity and a float for timestep.
* It then starts by decreasing the RigidBody’s velocity by itself multiplied by m\_linearDrag and timestep, simulating friction.
* The RigidBody’s position is then increased by m\_velocity multiplied by timestep, simulating movement.
* The RigidBody’s orientation is then increased by m\_angularVelocity multiplied by timestep, simulating rotation.
* The ApplyForce function is then called, taking in gravity multiplied by m\_mass and timestep to simulate the force of gravity, and (0, 0) for the position on the RigidBody that the force is applied to.
* The RigidBody’s angular velocity is then decreased by itself multiplied by m\_angularDrag and timestep, simulating rotational friction.
* If the affected RigidBody is a trigger, then for all PhysicsObjects in the vector m\_objectsInside, if they’re at the end of m\_objectsInsideThisFrame, a vector of PhysicsObjects, then if triggerExit is true, triggerExit is called on a pointer to the current PhysicsObject, and the current PhysicsObject is erased from m\_objectsInside.
* If the current PhysicsObject is also at the end of m\_objectsInside, the for loop ends. m\_objectsInsideThisFrame is then cleared.
* If m\_velocity and m\_angularVelocity are below the minimum thresholds, they get set to 0. This stops them from moving forever.

Collision Loop

* The collision loop starts inside PhysicsScenes’ Update function.
* It first iterates through m\_actors minus 1 using an iterator called outer starting at 0.
* Inside that for loop, another for loop starts that iterates through m\_actors using an iterator called inner that starts at outer + 1.
* This ensures that every actor inside m\_actors is checked against every other actor inside m\_actors.
* Inside that for loop, 2 PhysicsObjects are made: object 1, defined as the actor at outer’s position, and object 2, defined as the actor at inner’s position.
* Two integers are then defined, shapeId1 as the shape type of object1, and shapeId2 as the shape type of object2.
* Another integer called functionIdx is then defined as shapeId1 multiplied by SHAPE\_COUNT, plus shapeId2.
* A function called collisionFunctionPtr is then defined as the function at the position of functionIdx in collisionFunctionArray.
* If collisionFunctionPtr is not null, collisionFunctionPtr gets called using object1 and object2 as parameters.

Management of PhysicsObjects

PhysicsObjects are managed in a vector of pointers called m\_actors. This allows them to be easily accessed, added, and removed from the PhysicsScene.

Collision Callbacks

collisionCallback is a function that can be used to check for triggers when 2 actors collide. This allows for more reactions to be added to apps made using the engine.

Third Party Libraries

* Third party libraries used in the physics sim include GLM and AIE’s Bootstrap.
* GLM provides various math functions and variables. It was often used for the vector variables in this program.
* AIE’s Bootstrap provides basic rendering and input functions. This was used for displaying shapes, text, and providing input for testing and visualising the simulation.

Potential Improvements

The collision between PhysicsObjects could be made more resilient, as often when testing the engine, a strong enough force could cause objects to pass through each other.

A screenshot of a computer program

AI-generated content may be incorrect.