Physics/AI Boids Flocking Demonstration

I used the provided header files to provide me with an idea of how the system is to be implemented. I did change them a bit too.

Classes

Boid

The Boid class stores and calculates everything regarding an individual Boid. This includes its RigidBody and methods that calculate its direction vector and up vector. It also has methods to check if a Boid or Obstacle is in view, then stores that in its corresponding std::vector. These std::vectors are used to calculate the forces to apply to the Boid body. There is also a bounding sphere of set radius, where if the Boid gets too close a torque and a force towards origin is applied to move it away from the boundary.

The header file contains a lot of constants which can be tweaked to provide a more realistic simulation. These constants are used throughout the Boid class, mostly in the force calculations.

Obstacle

I simplified the provided header files for Obstacles a lot. I made the decision of using spheres because they're more convenient to do calculations on. This led to the simplification of MiscObstacles.h and Obstacle.h into a single Obstacle class.

This obstacle class stores its RigidBody, as well as its radius. These are used in the Boids' force calculations, so they know exactly where the obstacles are and can have forces applied to avoid them.

Flock

The Flock class is another simple class. This just stores all Boids and Obstacles in the scene. It has an Update method that loops through all Boids and have them check everything else in the scene for visibility. It then loops through the Boids again and this time have them apply forces according to what they can see. This calculation is done in the Boid class, and is separate from all other Boids.

Main Program

This is the main bullet program that was provided in the template. I removed the floor and added my scene, which consists of 50 Boids and 20 spheres as Obstacles. There was an error which caused Boids to just disappear after a while, so I had to disable deactivation to keep them in the scene. These Boids and Obstacles are added to the Flock std::vectors that stores the scene for future use. The Boids were also given a random initial linear velocity, otherwise some functions will contain a divide by 0 error, which crashes the program.

To create a better simulation, both the positions of the Boids and obstacles are randomly generated within level bounds. This is done by continually generating vectors until it is within level bounds, then that position is used.

I also slightly modified the shootbox() method to shoot boxes in straight lines. This allows for much easier testing of flocking.

Methods

Boid Class

Heading() | UpVec()

These methods calculate the Boid's direction of travel and "up" vector respectively. The up vector is used for rotation calculations which apply torque to the Boid. The direction of travel is used in angle calculations and the direction to apply thrust to, which moves the Boid "forward".

canSee(Boid *boid) | canSee(Obstacle *obstacle)

These methods check if another Boid or Obstacle is in "view", and if they are visible then they are added to the corresponding std::vector for later calculations. The way this is achieved in the same way for both Boids and Obstacles. Its distance and angle from the Boid in question is calculated, and compared to the visible distance and angle constants.

Update()

This is a simple method that provides an interface for the Flock class to tell the Boid to do everything. It calls the following four internal functions detailed below.

CheckBoundary()

This is the method that checks the position of the Boid relative to the boundary "sphere". Since the boundary sphere is centred on the origin, it just needs to compare the length of its position with the level radius, which is one of the constants defined in the header file. If the Boid is facing away from the centre, a torque is applied until it is no longer facing outwards or until it is within the bounding sphere.

ApplyPhysicalForces()

This applies standard forces to the Boid: thrust, lift and drag. To calculate the thrust force, I needed to get the world transform of the Boid, then calculate the thrust in local coordinates. A torque is also applied to rotate if necessary. For lift calculations, I calculated the TNB frame of the Boid and applied the force according to the B vector. For drag, I just reversed the linear and angular velocity and multiplied them by their coefficients. Linear drag is applied as a standard force, while angular drag is applied as torque.

ApplyFlockingForces()

After initialising the various vectors to 0, I can start adding to them in a for loop. The std::vector of visible Boids is looped through, calculating the position difference as well as the direction difference. Position difference is added to the cohesion vector, the negative of position difference added to the separation vector, and direction difference added to the alignment vector. After adding the values from all the visible Boids, it is then averaged in order to keep things the same no matter how many Boids are visible.

These forces are then applied after being multiplied by their coefficient. Torque is also applied to have the Boids point in their direction of travel and also have them level with each other. At the very end, the std::vector of visible Boids gets cleared so it can get filled again when checking for visible Boids on the next loop.

When there are no visible Boids, the current Boid is the 'leader'. A small random force is applied instead of all the flocking forces, to make create some movement for other Boids to follow.

ApplyObstacleForces()

This is fairly similar to the CheckBoundary() method stated above. It loops through all visible Obstacles, applying a force and torque to the Boid in each loop that makes the Boid avoid the Obstacle. It also calculates which way to turn, left or right, depending on the angle the Boid is approaching the Obstacle. This makes for more realistic dodging of Obstacles.

Obstacle Class

The Obstacle class only contains get functions for the body and radius.

Flock Class

addBoid(Boid* b) | addObstacle(Obstacle* o)

Used in the main program, these functions add Boid and Obstacle pointers to their corresponding std::vector.

Update()

This method loops through all Boid and have them check for visible Boids and Obstacles, then call their Update method which applies the forces.

Optimisations

I started development without using pointers and instead pass everything by value. This led to a huge overhead, RAM usage was going up around 100MB per second until it crashed. It took a while to figure out, but after changing all std::vectors to store pointers to objects instead of the object itself, it made the program run much faster and smoother, and removed the huge RAM usage.

Issues

Some Boids seem to not get affected by any forces and just go on indefinitely. Tried tweaking all parameters, couldn't get it fixed