Computing for Graphics:

Flocking System – Algorithm implementation

By Kristiyan Aladjov

# Introduction

For my Computer for Graphics assignment I developed a program that visualizes a three-dimensional Craig Reynolds’s flocking simulation that aims to recreate the movement of birds, fish or other animal species that exhibit grouping behaviours. The system is created using the NCCA Graphics Library (NGL) and is heavily based on Daniel Shiffman’s interpretation and implementation of Craig Reynolds’s found in his book “The Nature of Code”. The general idea is that system consists of independent agents known also as boids that move based on internal forces such as a desire to reach a certain position or avoid an obstacle or threat. These could be examined as physical forces but in the context of this program they are regarded as a natural behaviors that drive the agents to perform certain tasks. A flocking simulation is a result of a combination of several such behaviours that are triggered once two or more boids come into contact with each other.

The program also allows the user to manipulate certain values during run-time through a graphical user interface (GUI) which affects the resulting movement. The GUI was constructed using Qt5 and the tools found in Qt Creator.

# Overall structure

The way my program is structured in terms of classes begins with the MainWindow class which simply contains an instance of a GUI object and a NGL viewport (NGLScene) and allows the exchange of information between the two using slots and signals. The NGLScene class is the viewport in which the actual simulation and any other three-dimensional graphic objects are drawn. The simulation is actually made within the Flock class which is instanced in the NGLScene. The Flock class is used for managing the flock’s size, attributes and behaviour weighting. It acts a mediator between the scene and the actual boids. Lastly the boid class deals with the calculation of the position, velocity and behaviours.

## NGLScene class details

The NGLClass is the receiver for all the signals from the GUI but most of the time is just passes down that information to the Flock class. Apart from that it deals with the following operation:

* initializing the shader programs that all of the object in the scene will be drawn by
* set up the viewport camera
* calculating the model transformations based on inputs from the mouse
* initialize and draw the bounding box that contains the simulation

## 2.2 Flock class details

The Flock class is where boids are created or removed from the simulation. That is done by keeping all boids in a dynamic array (std::vector) and then adding or removing elements during run time.

The Flock class is responsible for setting up the attributes of the boids. These attributes or more accurately limitations are the maximum speed that the boids can move at, the maximum amount of force that can be applied to their steering and the radius within which boids become aware of each other. These attributes are received from the NGLScene class in the form of a std::map that contain float values and use strings as keys.

The last operation that this class has to perform before drawing the boids is weighting the boids’ behaviours. That is again done by getting a std::map containing a weight value of type float for every behvaiour and then multiplying the calculated vector for every behaviour by its corresponding weight.

## 2.3 Boid class details

Most of what the boid class does is perform the calculations for each behaviour which is described below in more detail. The way that the boid’s position is updated based on these behaviours is by following Newton’s 2nd Law of Motion assuming that the boids have a mass equal to one. All of the weighted internal forces are added together and assigned to the acceleration which is then added to the current velocity which in turn is added to current position of the boid. Once this process is done, the acceleration is set to 0 and everything is recalculated on the following update of the viewport.

# Boid behaviours

## Steering

Steering is at the core of every behavior. Steering itself is not a behavior but a calculation of the boid’s movement that takes into account its current velocity and the velocity it would need to have in order to reach its position of interest. Without steering the boid would immediately assume the velocity directed at its desired location but would lack a sense of realistic movement. Steering lets the boid transition

## Behvaiours in isolation

Each boid has a radius within which it becomes aware of other boids and begins flocking with them. While there are no other boids within that radius the boid has two possible options – it can either seek a goal object if one exists or wander without any general goal. Wandering is done by simply giving the boid a random target position which changes periodically based on an integer value that increments every time the viewport is updated. The wandering behaviour is immediately nullified once other boids are encountered although the changes it applied to the velocity of boids echo through the calculation of the group behaviours due to the steering force.

## Group behaviours

Once two or more boids become aware of each other their group behvaiours come into play. These behvaiours or rules as their often referred to are cohesion which causes the boids to come together at a center relative to their positions; separation which forces the boids to avoid each other and alignment which forces them to move in the same general direction.

All three rules are implemented in a similar way. Each boid would compare its position to every other boid’s position in a flock array. If the distance between them is smaller than a user-defined radius it takes a positon, position difference or velocity vector, depending on which rule is being calculated and adds it to a sum which is then averaged out by dividing the final sum by the total amount of neighbours. The resulting value is then ran through the steering force calculation and stored in a three-dimensional vector.

# Graphical User Interface

# 