Starling Murmuration

G.Sai Dheeraj(2016CS10349)
Danam Harshith Chandra(2016CS10364)

Indian Institute of Technology Delhi

1 Introduction

The starlings are generally a highly social family. Most species associate in flocks of varying sizes throughout the year. A flock of starlings is called a *murmuration*. We will simulate the movements of the murmaration with the help of boids.Boids is an artificial life program, developed by Craig Reynolds in 1986, which simulates the flocking behaviour of birds.The name "boid" corresponds to a shortened version of "bird-oid object", which refers to a bird-like object.As with most artificial life simulations, Boids is an example of emergent behavior; that is, the complexity of Boids arises from the interaction of individual agents (the boids, in this case) adhering to a set of simple rules. The rules applied in the simplest Boids world are as follows:

- **Separation**: steer to avoid crowding local flockmates
- **Alignment**: steer towards the average heading of local flockmates
- **Cohesion**: steer to move toward the average position (center of mass) of local flockmates

Apart from these basic rules we are applying two more rules to model the starling boids, and they are as follows:

- **Velocity Boundary**: limiting the maximum and minimum velocity of the boids
- **Position Boundary**: limiting the spacial boundary of the boids
- **Fixed Points**: these are objects which the starlings avoid in their movement

We will use these rules to simulate the famous starling murmuring movement observed in nature

2 Input Specifications

The user can change the visual by changing the number of boids in a simulation and they can also influence how the boids move inside the murmuration. Their input will be in given using predefined control buttons.

3 Mathematical Formulation

We assume each boid to be an individual body having two properties, ***velocity*** and ***position***. Both these properties determine how each of these starlings move. Consider the velocity of the an arbitrary boid b_i to be $v_i = (v_1, v_2, v_3)$ and its position to be (x_i, y_i, z_i) . At the start of the simulation both the position and the velocity will be initialized with a random value, that way when the simulation starts they all fly in towards the middle of the screen, rather than suddenly appearing in mid-air. Firstly, we look at how the velocity of the boids change followed by the change in position.

3.1 Velocity Transformation

The velocity of the boids are affected individually by four rules which have already been mentioned above. Each rule is a essentially a function that is applied to the boid, let they be named *Rule1*, *Rule2*, *Rule3*, *Rule4*.

$$u_1 = Rule1(b_i)$$

$$u_2 = Rule2(b_i)$$

$$u_3 = Rule3(b_i)$$

$$v_i = v_i + u_1 + u_2 + u_3$$

Each of these rules apply independently and the final velocity is a resultant of the application of all these rules. We will discuss about each of these rules in detail.

Rule1: **Separation**
This rule states that *boids try to keep a small distance away from other objects including other boids*. This is to ensure that they don't collide with each other.We look at each boid, and if it's within a defined small distance (say 100 units) of another boid move it as far away again as it already is. This is done by subtracting from a vector c the displacement of each boid which is near by. We initialize c to zero as we want this rule to give us a vector which when added to the current position moves a boid away from those near it. For each boid calculate the following if $(|b_i.position - b_j.position| < r)$, where r is least possible distance

$$c = c - (b_i.position - b_j.position)$$

After doing this cumulatively for all the boids which are close to the boid, we get the result.

$$u_1 = c$$

Doubling the distance ensures that boids which are close by are not repelled immediately, which gives the effect of bouncing, rather this would provide acceleration in the opposite direction. They will be slightly steered away from each other, and at the next time step if they are still near each other they will be pushed further apart. Hence, the resultant repulsion takes the form of a smooth acceleration. If two boids are very close to each other it's probably because they have been flying very quickly towards each other, considering that their previous motion has also been restrained by this rule. Suddenly jerking them away from each other, such that they each have their motion reversed, would appear unnatural. Instead, we have them slow down and accelerate away from each other until they are far enough apart for our liking.

Rule2: **Cohesion** This rule states that *boids try to fly towards center of mass of other boids.*The 'centre of mass' is simply the average position of all the boids. We use the term centre of mass by analogy with the corresponding physical formula however, we ignore individual masses here and treat all boids having the same mass. Assume we have N boids, called b1, b2, ..., bN. Also, the position of a boid b is denoted b.position. Then the 'centre of mass' is calculated with of the remaining $N-1$ boids given by:

$$pc_j = (b_1.position + b_2.position + .. + b_{i-1}.position + b_{i+1}.position + .. + b_N.position)/N-1$$

Having calculated the perceived centre, we need to work out how to move the boid towards it. To move it 1 of the way towards the centre (this is an example) this is given by $(pcJ - bJ.position)/100$.Thus we have calculated the first vector offset, u_2, for the boid.

Rule3: **Alignment**
This rule states that *boids try to match velocity with near boids.*This is similar to Rule1, however instead of averaging the positions of the other boids we average the velocities. We calculate a 'perceived velocity', pv_j, then add a small portion to the boid's current velocity.

$$pv_j = (b_1.velocity + b_2.velocity + .. + b_{i-1}.velocity + b_{i+1}.velocity + .. + b_N.velocity)/N-1$$

$$u_3 = (pv_j - b_i.velocity)/8$$

Rule4: **Fixed Points**
This rule is utilized when boids approach objects called as fixed points. In such cases the boids receive velocity in the reverse direction proportional to the proximity of the boid with respect to the object.

Rule5: **Velocity Boundary**
This rule is states that *the boids have a limitations on their velocity.* This rule is to ensure that none of the boids have too high or too low velocities. This rule is only applied in the above mentioned two cases. We will reduce the velocity by half in the first case and double it in the second case. The velocity direction remains unchanged in either of them.

Rule6: **Position Boundary**
This rule states that *the boids have a limitation on their position.* When any boid moves outside the defined boundary we add a velocity proportional to the distance travelled outside the bondary in the opposite direction. This is to ensure that there is a smooth transition of the boids and they don't have sudden erratic movements.
Rule4, Rule5, and Rule6 only get applied in the niche cases mentioned and in general the velocity is solely determined by Rule1, Rule2, and Rule3.
The final velocity of boid b_i is given by:

$$b_i.velocity = b_i.velcocity + u_1 + u_2 + u_3$$

3.2 Position Transformation

We interpret the position as proportional to the distance moved by the boid, hence the position change will be proportional to the velocity. To find the change in position we add the new velocity of the boid to its initial velocity.

$$b_i.position = b_i.position + b_i.velocity$$

4 Ending Remarks

Using these rules we can simulate to a certain extent the behaviour of these birds. The simulation can be subject to some minor changes due to implementational issues.