IIT Delhi

COP290

Starling Murmuration

Modeling and Analysis

Submitted By:
Madhur Singal
2015EE10908

Garvit Gupta
2015EE10447

Yashti Agrawal
2015PH10840

April 25, 2018

1 Flock Motion

1.1 Introduction

Flocking behavior is the behavior exhibited when a group of birds, called a flock, are foraging or in flight. At sunset the birds take flight to search for a roost, creating immense flocks that darken the sky and blot out the sunset. This flocking behavior also protects them from predators as the predators get confused.

1.2 Initialising Boids

The simulation should start with n boids initially, where all the boids should start at **unique random** positions on the screen with random velocities. The unique positions property is important to maintain because 2 boids at the same position would mean a collision which is not the case in a practical system.
To ensure that the initial positions are random, we can use a Random Number Generator as follows: Suppose we want to generate n random numbers in range [a,b). We can first find random integers in range [0,2n) and then scale them to range [a,b). If r_I is random integer, then it's corresponding random number r required would be :

$$r = \frac{r_I * (b-a)}{2n} + a$$

To find random integers in range [0,2n):

1. Make an array arr of size $2n$ such that $arr[i] = i$.
2. Start with $i = 2n - 1$
3. Generate a random integer j from [0,i).
4. If $arr[j]$ is equal to j, replace $arr[j]$ by i and $arr[i]$ by j, decrement i by 1 and repeat step 3. Else, repeat step 3 with same i.
5. Keep on doing the above steps till i doesn't become equal to $n - 1$.
6. The numbers from $arr[n]$ to $arr[n+1]$ are the required random integers.

This is a very optimum method of computing random numbers because checking if a random number has already occured takes only $O(1)$ time.

1.3 Dependencies

Each starling in a murmuration coordinates it's speed and position with it's **7 nearest neighbours**. The complexity of the flocking behaviour arises from individual interactions adhering to a set of simple rules:

1. **Alignment:** Steer towards the average heading of local flockmates
2. **Cohesion:** Steer to move toward the average position (center of mass) of local flockmates
3. **Separation:** Steer to avoid crowding local flockmates

1.3.1 Alignment

Boids try to match the velocity with neighbouring boids, i.e. with the velocity of the Centre of Mass of the 7 nearest neighbours. Let $v_1, v_2, ...v_7$ be velocities of the 7 neighbours of boid b with velocity v.
Then, the velocity of centre of mass of neighbours is given by:

$$c = \frac{(v_1+v_2+....+v_7)}{7}$$

The velocity update for boid b is given by:

$$v_1 = c - v$$

1.3.2 Cohesion

Boids try to fly towards the Centre of Mass of 7 neighbouring boids.
Let $n_1, n_2, ...n_7$ be positions of the 7 neighbours of boid b with position p.
Then, the centre of mass of neighbours is given by:

$$c = \frac{(n_1+n_2+....+n_7)}{7}$$

The velocity update for boid b is given by:

$$v_2 = c - p$$

1.3.3 Separation

Boids don't collide with each other and try to maintain a minimum distance between the neighbouring boids. For this, find those boids that are very close to the current boid b (at a distance less than or equal to the minimum distance to be maintained) and move the boid b away from each of the very close boids.
For this, the following algorithm is followed:

1. Take a vector c and initialise it to 0.
2. For each boid b_i other than b, if distance between b and b_i is less than the minimum distance to be maintained, $c = c - (b_i.position - b.position)$
3. The velocity update for boid b is given by: $v_3 = c$

1.4 Updating Boids

The velocities of boids will have to updated according to the rules in Section 1.3 and a corresponding change in position will have to be made according to the time interval for all the boids.

1.4.1 Velocity Update

For each boid, the velocity updates obtained from the 3 rules of Section 1.3 have to be scaled appropriately and added to the current velocity to get the new velocity. So, the scaling constants for these updates become the hyperparameters of our system that will be tuned appropriately during the simulation.

For each boid:

$$velocity = previous\ velocity + c_1.v_1 + c_2.v_2 + c_3.v_3$$

where c_1, c_2 and c_3 are hyperparameters.

1.4.2 Position update

For each boid, the position would have to be updated by adding the product of calculated velocity and the time interval between the position update to

the previous position.

For each boid:

position= previous position + velocity(time interval)*

1.5 Constraints

There are some other constraints that need to be taken care of like Boids don't go outside the boundaries of the screen and don't fly at unusually high speeds.

1.5.1 Boundary Conditions

When a boid approaches near the boundary, it's velocity component perpendicular to the boundary is slowly turned away from the boundary, by applying a force away from boundary, inversely proportional to distance from boundary. The proportionality constant is a hyperparameter.

1.5.2 Limit Velocity

The additions of velocities scaled appropriately may add up to give a velocity that is large. The velocities need to be normalized in such a case.
If the magnitude of calculated velocity comes out to be greater than *max speed*, the velocity is scaled down to have magnitude equal to *max speed*. Here, *max speed* is a hyperparamter.

2 External Factors

2.1 Introduction

In a real setting, the flocking behaviour is influenced by many factors like the presence of Obstacles and Predators. Boids will try to avoid the obstacles during their flight and also run away from Predator.

2.2 Predator

There are 2 things to be considered in the presence of Predators:

- Motion of Predator
- Effect on motion of Boid

2.2.1 Motion of Predator

Predators move in such a way that they try to chase the nearest boid. Rarely, a predator may also choose to give weightage to moving towards the flock, but that is not a good strategy for Predator. Chasing the overall flock won't be beneficial for the Predator and it will end up getting confused.

- **Velocity Update:**
  For each predator p having the nearest boid b:

  $$velocity = previous\ velocity + c_4.(b.position - p.position)$$

  where c_4 is a hyperparameter

- **Position Update:**
  For each predator, the position would have to be updated by adding the product of calculated velocity and the time interval between the position update to the previous position.

  For each predator:

  $$position = previous\ position + velocity*(time\ interval)$$

2.2.2 Effect on motion of Boid

Boids want to run away from the Predators and thus try to maintain a minimum distance from them. For this, find those predators that are very close to the current boid b (at a distance less than or equal to the minimum distance to be maintained) and move the boid b away from each of the very close predators.
For this, the following algorithm is followed:

- **Velocity Update:**

  1. Take a vector c and initialise it to 0.
  2. For each predator p_i, if distance between b and p_i is less than the minimum distance to be maintained, $c = c - (p_i.position - b.position)$
  3. The velocity update for boid b is given by: $v_5 = c$

  For each boid:

  $$velocity = previous\ velocity + c_1.v_1 + c_2.v_2 + c_3.v_3 + c_5.v_5$$

- **Position Update:**
  For each boid, the position would have to be updated by adding the product of calculated velocity and the time interval between the position update to the previous position.

  For each boid:

  $$position = previous\ position + velocity*(time\ interval)$$

2.3 Obstacle

Boids will try to avoid colliding with the obtacles while flying. For this, find those obstacles that are very close to the current boid b (at a distance less than or equal to the minimum distance to be maintained) and move the boid b away from each of the very close obstacles.
For this, the following algorithm is followed:

- **Velocity Update:**

  1. Take a vector c and initialise it to 0.
  2. For each predator p_i, if distance between b and p_i is less than the minimum distance to be maintained, $c = c - (p_i.position - b.position)$

3. The velocity update for boid b is given by: $v_6 = c$

For each boid:

$$velocity = previous\ velocity + c_1.v_1 + c_2.v_2 + c_3.v_3 + c_5.v_5 + c_6.v_6$$

- **Position Update:**
  For each boid, the position would have to be updated by adding the product of calculated velocity and the time interval between the position update to the previous position.

  For each boid:

$$position = previous\ position + velocity*(time\ interval)$$

3 References

1. "Boids Background and Update" by Craig Reynolds.
2. http://cp3-origins.dk/a/14187
3. http://www.pnas.org/content/111/29/10422