AI Steering Behaviors: Flocking

Jesse Harrison

jesse.harrison@digipen.edu

This article takes a look into one of the most popular, if not most popular, steering behaviors that exists today, Flocking. It will cover the beginnings of this fascinating artificial behavior. It will also cover basic implementation of the base of the algorithm and possible improvement, taking into consideration algorithm complexity.

Basic Flocking

Everyone has an intuitive idea of what flocking looks like, we see it in nature among such animals as birds, fish and sheep, and it appears to be this extremely complex all knowing single entity, where every animal is aware of every other animals positions, future positions and possibly even thoughts. How could this ever be performed programmatically and in real time no less.

Craig Reynolds was the first person to simulate this behavior on a computer, which has earned him the unofficial title as the father of flocking. His first simulation took place back in 1986 and the ingenious basic structure he used still holds strong to this day. Craig proposed that flocking was an emergent behavior, that is each member or agent of a flock followed simple rules in order to create this group behavior. Craig also gave an interesting name for each member of the flock, he called them boids. So, in order to create the group behavior of flocking each boid would follow three simple rules: separation, alignment, and cohesion. Separation prevents collision of neighboring boids, alignment promotes direction uniformity between boid neighbors and cohesion keeps neighboring boids together. Now you may have noticed what all three of these rules have in common, neighboring boids are always in consideration.

Neighbors

Each boid in our flock must keep track of their neighbors in order to avoid colliding, remain together, and move in the same direction, but how does each boid keep track of their neighbors. The most common is the brute force way where each boid has a specified radius, we'll call this the neighbor radius. It is safe to say we need some sort of container in which we keep all of our boids. So, every frame each individual boid would loop through this container of boids getting the distance between itself and a boid in the container if one of those distances is less than the boids neighbor radius then we know that it is a neighbor. Please refer to Figure 1. It is easy to see that this will start to get computationally expensive the more boids that you have, and in fact this is the brunt of the flocking algorithm. Their is, however a quick and easy optimization that can be performed here. As we know distance calculations are very expensive simply because of the nefarious square root operation. To avoid using n^2 amount of square root operations every frame we can simply compare the distance squared between the boids and the neighbor radius squared, since we don't actually need to know the distance between boids.

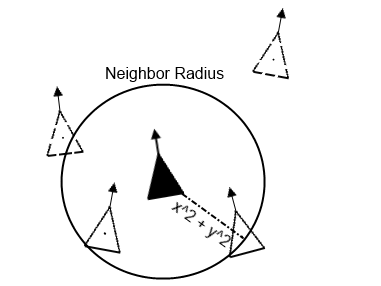


Figure 1. Neighbors of boid based on distance between boids and the neighbor radius. Dotted boids are not neighbors of the boid in question(solid black boid).

Separation

This rule is in place in order to avoid collisions between boids. Each boid will now have a second radius surrounding it, lets call it their separation radius, convenient isn't it. If any boids neighbors enter into this zone then the boid who's zone has been violated steers away from the intrusion in order to avoid collision. Please refer to Figure 2. To detect an intrusion simply use the same method we used early for the neighbors, calculate the square distance between the neighbors and the current boid and if it is less then the separation radius squared an intrusion has occurred and the boid must steer away.

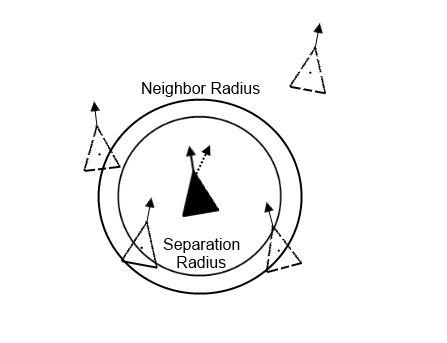


Figure 2. Separate boids based on separation radius, dotted boids not intruding on radius.

The dotted direction of the boid in question(the solid black boid) is the new direction in which the boid will steer.

Alignment

This rule is in place so that all of the boids head in the same general direction. To do this we simply have each boid steer in the averaged direction of their neighbors. One way to perform this would be to take each boids velocity vector and normalize it to get its direction vector. Then given all of the neighbors direction vectors take the average and divide it by the number of neighbors, this will give you a vector that is the average direction, normalize this new averaged direction vector and have the specified boid steer in that direction.

Cohesion

This rule is in place so that boids will stay close together in order to form a group. The approach for this is very similar to alignment except instead of the neighbors direction we consider position. So this is accomplished by having each boid steer themselves toward the average position of their neighbors. This could be done simply by obtaining the vector sum of the neighbors respective positions and dividing it by the number of neighbors, the result is the average position in the form of a vector, which the boid will steer towards. Given this rule note the importance of the separation rule, without it all of our boids would form into a single boid.

All of these rules together create an extremely accurate depiction of flocking. However, there are some additional rules that can be added.

Additional Rules

In addition to the three basic rules that comprise the flocking algorithm, there exist some common rules that tend to be added to amplify the realism of a flock. The rules consist of: changing goals, obstacle avoidance, and wandering.

Changing goals

There is not a whole lot that can be said about the implementation of changing goals because it is quite simple. That being said this is can be a very important rule to have in a game if your game uses flocking. For example in an RTS game if you had a group of units that need to get from one point to the next you would need a specific goal. As is with our current three simple rules our flock has no goal, they just meander about. To set a goal for our flock we would first need specify a point in the environment a goal that each boid knows about. Then simply have each boid steer in the direction of the goal, its that easy, our three other rules will take care of the rest. Keep in mind if the flock has a leader then only the leader would need to know about the goal.

Obstacle Avoidance

I think it is pretty obvious how important this additional rule can be, whether in a basic bird flocking demonstration or in a game, it wouldn't look right if the flock began magically moving through objects. This rule is a little more difficult than goal changing, it requires some basic intersection tests that would be a little more difficult in 3D then 2D, however the general idea applies to both 2D and 3D. For obstacle avoidance each boid is given a virtual feeler that sticks out of the front of it. The feeler will do just that feel for upcoming obstacles and avoid them. The feeler isn't difficult to add since the boids direction vector is already know we take that and multiply by some constant to have it stick out a little longer, this length is somewhat trial and error and depends on the boids speed, if the boids speed is too fast and the feeler is too small the boid wont be able to steer away in time, so feeler size will take some adjusting. Feelers feeling objects refers, of course, to intersection. Assuming that the feeler is represented as a ray, ray intersection tests need to be in place, that is, ray to whatever objects you wish to avoid. You must steer away enough to avoid the object, so information about the object itself, such as its width is needed. As you can see obstacle avoidance gets more difficult with non-uniform objects. A simple and easy trick would be to have all of the desired collision objects surrounded by axis aligned bounding boxes, which would result in a single intersection test and also provide a uniform length in which to avoid. However, the movement will not end up looking as smooth. Also, additional feelers can be added for more in depth information, such as if an enemy is attacking from the side of the unit and we want to detect this the same type of feeler would be added but sticking out of the side of the boid as opposed to the front. This can be thought as peripheral vision.

Wandering

Wandering can be an important addition for a flock to have in a game, it can provide an interesting illusion of intelligence. For wandering we provide each boid with a basic variance of its current direction in 2D this can be done by putting an invisible circle in front of each boid, for 3D a sphere, and have the direction in which the boid moves be based off of the edge of the circle, providing a smooth back and forth motion. Note that that the boid heads in the same general direction wandering will just have the boid avoid moving in a straight line.

Algorithm Complexity

The basic implementation of the flocking algorithm is O(n^2). This is due to having to find each boids neighbor each frame. Each boid has to look at every other boid. A possible improvement would be to use bin-lattice spatial subdivision. This would require that the flock only has a predetermined area in which it could move. The allot-ed area in which the flock can move is divided into bins. A bin knows which boids it stores which allows for quicker neighbor checks based only off the current bin of the current boid. The lattice would need to be updated only when a boid swaps bins, which isn't terrible. The complexity of this comes out to be O(n\*k), with n being the number of boids and k being the number of bins. This is a definite improvement, but can become detrimental if too may bins are created. If more bins are created than there are boids the algorithms complexity is now greater than O(n^2).

Conclusion

With the basics of flocking, that is separation, alignment, and cohesion, an amazingly realistic flock can be created. So, if you are interested in implementing flocking for yourself don't hesitate, it is really quite easy. The typical application for flocking in games is a RTS style game where entire units are meant to be moved simultaneously, but there are so much more applications that can be considered. Flocking has been used in squad based games, jet fighting games, and even movies. In my opinion it is one of the most interesting AI behavior's and is definitely worth looking into if your interested in AI, emergent behavior, or even nature.

References

Book:

[Bourg04] Bourg, David M., and Glenn Seemann. *AI for Game Developers*, O'Reilly Media, Inc., 2004. 52-79.

Web source:

[Reynolds95] Reynolds, Craig, “Boids Background and Update,” available online at http://www.red3d.com/cwr/boids/, June 25, 1995.