CONSTRAINED 3D FLOCKING BEHAVIOR*

Greggory Hernandez
Curtis Welborn
Computer Science Department
Utah Valley University
Orem, Utah 84058
801 863-7058
greggory.hz@gmail.com
Curtis.Welborn@uvu.edu

ABSTRACT

Craig Reynolds introduced the concepts of simulated flocking behavior via the use of simple aggregate Boid behaviors which make the Boids appear to exercise independent autonomous actions. The term Boid is used to identify the actors within the flock without attaching any particular animal characteristics with the actor (e.g., fish, bird, mammal). Artificial potential-fields (APF) are often used to implement collision avoidance for robotic and multi-agent systems, making them well suited for implementing flocking behaviors. AFP can also be used to define constraints that direct the motion of a flock while still allowing for simple aggregate Boid behaviors. Flocking behavior is used in computer games or movies where groups of Boids are to move together without appearing to move in lockstep. This paper explores the approach used for directing the flocking behavior of Boids via APF constraints. The APF constraints are intended for an end user (e.g., animator or director) inexperienced with programming or the technical details of implementing flocking behavior or APFs. Our constraint system for controlling the motion of the flock is currently composed of two constraint types:

A fly-by waypoint constraint is used to control the direction of the flock in a manner similar to the scripted flocking defined by Reynolds. A user defines 3D waypoints on the screen to map a sequential path that the flock is to follow.

---

* Copyright © 2011 by the Consortium for Computing Sciences in Colleges. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the CCSC copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Consortium for Computing Sciences in Colleges. To copy otherwise, or to republish, requires a fee and/or specific permission.

A boundary constraint is used to control the outward spread of the flock. A user defines a boundary between waypoints that sets the maximum outward spread of the flock by limiting the flock to move within the channel/conduit defined between the waypoints.

INTRODUCTION

Craig Reynolds [2] introduced the concepts of simulated flocking behavior via the use of simple aggregate Boid behaviors which make the Boids appear to exercise independent autonomous actions. The term Boid is used to identify the actors within the flock without attaching any particular animal characteristics with the actor (e.g., fish, bird, mammal). Artificial potential-fields (AFP) are often used to implement collision avoidance for robotic and multi-agent systems, making them well suited for implementing flocking behaviors. AFP can also be used to define constraints that direct the motion of a flock while still allowing for simple aggregate Boid behaviors. Flocking behavior is used in computer games or movies where groups of Boids are to move together without appearing to move in lockstep. This paper explores the approach used for directing the flocking behavior of Boids via APF constraints. The APF constraints are intended for an end user (e.g., animator or director) inexperienced with programming or the technical details of implementing flocking behavior or APFs. Our constraint system for controlling the motion of the flock is currently composed of 2 constraint types:

A fly-by waypoint constraint is used to control the direction of the flock in a manner similar to the scripted flocking defined by Reynolds. A user defines 3D waypoints on the screen to map a sequential path that the flock is to follow.

A boundary constraint is used to control the outward spread of the flock. A user defines a boundary between waypoints that sets the maximum outward spread of the flock by limiting the flock to move within the channel/conduit defined between the waypoints.

ARTIFICIAL POTENTIAL-FIELDS

Now that an introduction has been given involving some math for moving and orienting Boids in 3D space, a methodology is needed for making the Boids avoid collisions while being attracted to a flock. The methodology utilized here is Artificial Potential Fields [1] (aka Pfields). In a limited sense, Pfields is the application of vectors to the problem of robot navigation. In robot navigation every object, including the robot, is associated with one or more vectors. These vectors can be characterized as being either repulsive or attractive.

When looking at an object with a repulsive Pfield (Figure 1), the vectors are always directed away from the object. The magnitude of the vectors associated with the Pfield should become larger when you get closer to the object. As an analogy, consider standing in a room with someone you do not know and who looks a little creepy. When they are on the other side of the room, you just ignore them. If they move within about 5 feet, you will begin eyeing them; when they move within a foot, you might stick out your hand to keep them from bumping into you. As they move closer to you, you use a greater repulsive field to keep them away from you.

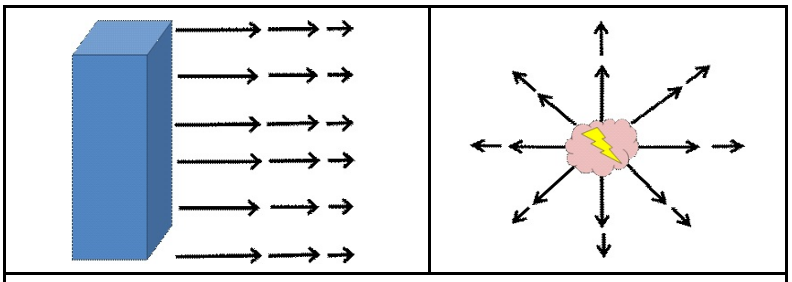

Figure 1. Repulsive Pfields.

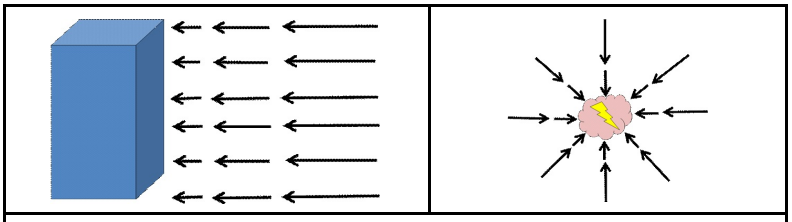

Figure 2. Attractive Pfields.

Attractive Pfields (Figure 2), always point toward the object, with the magnitude of the vectors becoming smaller as you get closer to the object. As an analogy, consider parking your car in your garage. When you turn onto your street, you are compelled to drive your car all the way to your house even if there is traffic on your street. However, as you get closer to the back wall of your garage, your desire to keep getting closer should diminish, unless you want to park inside the house.

Pfields will be associated with every object in the flocking simulation. Boids will have attractive Pfields between each other when they are within a specified distance to encourage their flocking. They will also have repulsive Pfields when they are close together to avoid them colliding with each other. Obstacles in the simulation will have repulsive Pfields that interact with the repulsive Pfields of the Boids to keep the Boids from colliding with the obstacles.

FLOCKING BEHAVIOR

In addition to the attractive and repulsive Pfields associated with every object, Boids also contain a single motion vector. A Boid's motion vector determines its heading and speed. To build upon the driving analogy from above, when you press the accelerator or brake pedal in your car, the car's motion vector will increase and decrease in magnitude, respectively. Turning the steering wheel would alter the direction that the motion vector is pointing. When a Boid enters the range of another Boid's attractive Pfields, the two Boids are pulled together to simulate flocking behavior. The best method to use to pull Boids together is to simply sum the two Boids' motion vectors and gradually apply the (normalized) resultant vector to each Boid.

This method makes the Boids' motion appear natural and smooth when the simulation is run. This works even when numerous Boids are in close proximity to each other since ultimately all the Boids' different motion vectors will converge on what is effectively a single average vector that all the interacting Boids will possess. As new

Boids come within range of the flock, the foreign Boid is assimilated by the motion vectors of the Boids nearest to it.

When the simulation is run, none of these interactions are under the control of the end-user and so the animation is entirely determined by the computer. This may not always be a desired situation. While it is appropriate to let the computer handle the more complex calculations, allowing the user some control can enhance the creative potential immensely.

USER CONSTRAINTS

With the implementation of flocking behavior and the entirety of the Boid simulation, there was always the glaring omission of user control. Up to this point, the Boid simulation that was implemented was entirely under control of the programmer. An end-user had no way of altering the parameters of the animation without being forced to dig into the code and tediously make changes in the hopes that it would be the change that was desired. In an attempt to remedy this, two constraints will now be introduced that the user will have direct control over: fly-by waypoints and boundary constraints.

Fly-by Waypoint

The first constraint available to a user is the Fly-by Waypoint constraint. This allows a user to input an arbitrary set of three-dimensional points (waypoints). When the user starts the simulation, a predefined number of Boids will be created (in a random configuration) around the first waypoint, and then the Boids will move as a flock to each waypoint in the order that the user entered them. This allows a user to create complex paths and more sophisticated animations without being required to have an intimate knowledge of the simulation's inner-workings.

In addition to the base flocking simulation described above, only two main constructs are needed to simulate fly-by waypoints: a list of waypoints, and a method for tracking the current waypoint. A method for allowing the user to input waypoints at runtime and a method for the user to start the simulation once all desired waypoints are entered is also required, but presently the focus will be on the implementation of the actual constraint rather than the details of exposing it to the user. It is much more important that the list of waypoints and tracking of the current waypoint be implemented correctly. Once that is accomplished, the details of a user interface is a trivial matter. All that is currently important to know is that a user is actually able to enter waypoints and can start the simulation. In addition, the simulation has two states: running and not running. The simulation always starts in the "not running" state and only enters the "running" state when instructed to do so by the user.

The list of waypoints is implemented as a circular first-in/first-out list. That is, a queue that goes back to the beginning of the queue upon reaching the final element. Each time the user inputs a new waypoint, that waypoint is added to the end of the queue.

The current waypoint initially refers to the waypoint at the front of the queue. When the Boids reach the current waypoint, that current waypoint will be changed to refer to the

next waypoint in the queue. If the current waypoint is the last waypoint in the queue, the current waypoint will be changed to refer again to the first waypoint in the queue; otherwise, the Boids would remain flocking around the last waypoint ad infinitum. In addition to tracking the current waypoint, each waypoint also has a state of "hit" or "not hit." Initially all waypoints are set to "not hit." Each time a waypoint is reached by the Boids, that waypoint is set to "hit." When the end of the queue is reached, all waypoints are reverted to "not hit." Among other things, this can allow for a visual queue to the user letting them know how far the Boids are along the path.

In this implementation, when the simulation begins, a user is able to click to place a waypoint in the x/y plane. Once it is placed, the user can then adjust the z value for that waypoint. When the user is satisfied, the user sends the "go" instruction to start the simulation.

Boundary Constraint

Depending on the configuration of the waypoints, Boids may end up with a large outward spread as they move between waypoints. They might also exhibit a sudden change in direction as the leading Boid reaches the waypoint first, and the current waypoint transitions to the next waypoint. These may not always be desired effects. To counteract this, a user is given the ability to set a maximum boundary in order to hold the Boids closer to the line created between the source and destination waypoints. With this constraint in place, Boids will move between waypoints without an overly large outward spread or odd behavior when changing directions.

A boundary constraint will be implemented that holds the Boids within a cylinder between the previous and current waypoint. The cylinder will have a user-defined radius. This radius is the maximum distance that a Boid is allowed to be from the line segment created between the two active waypoints. If any Boid reaches a distance from this line that is approximately equal to or greater than the cylinder's radius, it will be pulled back towards the nearest point on the line.

To compute the cylinder, the current waypoint and the previous waypoint are needed. If the current waypoint happens to be the first waypoint in the list, the last waypoint in the list is used as the previous waypoint. Once the previous and current waypoints are obtained, compute the axis-of-the-cylinder--a line that extends infinitely in both directions and passes through the previous waypoint and the current waypoint. After the axis-of-the-cylinder is found, the following process will be applied to each Boid in the simulation.

First compute the distance between a Boid and the axis-of-the-cylinder. To compute the distance, form a right triangle (see Figure 3) with the following points: the position of the current waypoint, the position of a Boid, and the point closest to the Boid on the axis-of-the-cylinder. Since the length of the hypotenuse is known (distance between the current waypoint and the Boid), and the angle (theta) can easily be derived between the hypotenuse and the segment between the current waypoint and the closest point on the axis-of-the-cylinder, the equation: dist = length of hypotenuse * sin(theta) can be used to find the Boid's distance from the line. If the distance is greater than the radius supplied by the user, then the Boid needs to be pulled back towards the center of that radius.

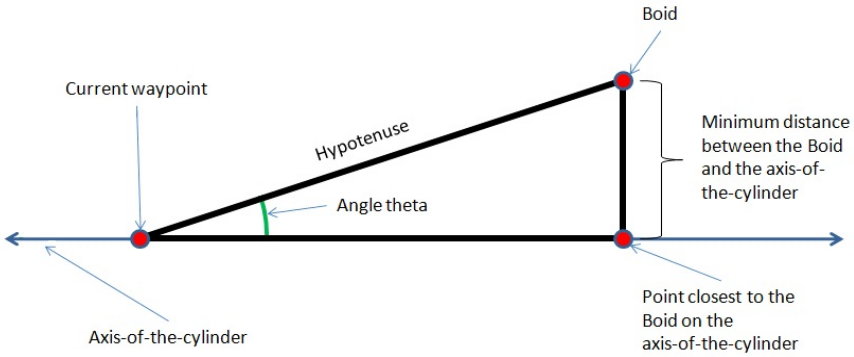

Figure 3. Distance between Boid and axis-of-the-cylinder.

While the Boid's motion can be altered using a vector that points from the Boid to the nearest point on the line, Boid movement is more natural and less abrupt if a vector is used that is a small distance further along the line segment nearer the current waypoint. Once this vector is determined, just sum the vector with the Boid's current motion vector to produce a new trajectory for the Boid that moves it toward the axis-of-the-cylinder.

With this method implemented, all Boids in the simulation will be forced to remain within the boundary constraint as defined by the user. While the primary purpose of this constraint is to allow the user more control over the details of the animation, as a secondary benefit this implementation of the boundary constraint also provides flocking behavior that is more akin to what is seen when a real-world flock turns a corner. The sequence of screenshots shown in Figures 4, 5 and 6 shows a flock of Boids following waypoints (red and blue dots) with a boundary constraint to confine their movement.

Figure 4. Boids moving toward read waypoint.

Figure 5. Boids turn after reaching blue waypoint.

Figure 6. Boids approaching red waypoint.

CONCLUSION

Adding the constraints of waypoints and boundaries adds a new level of sophistication to the original Boid flocking simulation. In addition, the end-user is allowed to have control over the simulation in ways that were not previously possible. With this extra layer of flexibility, higher quality animations become possible more easily by users with less experience.

Both the fly-by waypoints and the boundary constraints provide control without sacrificing the believability of the simulation in most cases. There are some notes to make about maintaining believability. With proper care, waypoints can provide a very

believable animation. However, the user should be aware that a more realistic simulation will usually have more fly-by waypoints in a complex configuration in order to avoid having paths that are predictable or that feel contrived. The user is, of course, free to create waypoints without restrictions (except perhaps those imposed by the computer on which the simulation is being run). With the boundary constraints, making the boundary too small can cause abhorrent behavior. The Boids will basically have nowhere to go so they end up jerking back and forth, often intersecting each other and remaining outside of the boundary. Making the boundary too small should be avoided. Experimentation is the best method to determine what works for any given simulation.

FUTURE WORK

As for the future of this work, active investigation is being done regarding the design and implementation of additional user constraints to control flocking behavior. While there is possibly an endless variation of constraints that could be manufactured, the additional constraints being considered can be roughly grouped into one of two categories of constraint types.

The first are constraints to restrict the shape of a moving flock. The boundary constraint defined in the paper is one example of a shape restricting constraint; additional 3-dimensional shape restrictions are being considered. When restricting the shape of the flock, it is important not to impose an unnatural laminar flow (characterized by smooth parallel changes) or turbulent flow (characterized by chaotic changes) on the flock.

The second are constraints that direct the movement of the flock as a whole. The fly-by waypoint constraint defined in the paper is one example of a flock movement restricting constraint. While the fly-by waypoint constraint allows a user to influence the direction of the flock, new constraints in this category will focus on influencing the speed of the flock.

REFERENCES

[1] Murphy, R.R., *Introduction to AI Robotics*, Cambridge, MA: MIT Press, 1998.

[2] Reynolds, C.W., Flocks, herds and schools: a distributed behavioral model, *Computer Graphics*, 21(4), 25-34, 1987.