CS312:Fall 2016

Assignment 3: 3D Flocking

Due Thursday 9/29, in groups of 1 or 2.

Overview: Implement a program that models the behavior of a 3D flock of synthetic birds, commonly known as *boids*. Boids are generally simulated as an emergent system. An important principle of emergent systems is the lack of central organization or leadership; it is a form of a spontaneous creation of order. Familiar examples from nature include shoaling of fish, swarming of insects and flocking of birds. All are considered emgergent behanviors because they arise from simple rules that are followed by individual group members and do not involve any central coordination.

Flocking motion are simulated via the following basic rules, developed in 1986 by Craig Reynolds:

- **Separation**: The boids should attempt to maintain a certain minimum separation distance from local flockmates to avoid overcrowding.
- **Alignment**: Move at roughly the same speed and steer towards the same heading as the local flockmates.
- **Cohesion**: Remain together as a group - move towards the average position (center of mass) of the local flock.

The above properties define the nature of the motion at local level, but do not constrain the group's overall motion. The animator (you) must be able to control the *global motion* of the boids as well. For example, the boids should move towards some goal point or follow a leader whose motion is presribed.

Program Requirements:

- **Local flocking behavior**: The motion exhibited by your flock should satisfy the three elements, separation, cohesion and alignment, as described above.
- **Number of boids**: The number of boids should be dynamically adjustable. Hitting the '+' key creates a new boid at a random location in the window and '-' removes a random boid from the scene.
- **Pause/Resume/Single Step**: Toggling the 'p' key pauses and resumes the simulation. Hitting the 'd' key pauses and single-steps the simulation. The '+' and '-' keys should work whether the program is paused or running.
- **Boid rendering**: When drawing a boid, do so in a manner that indicates the direction of flight.
- **Reshape**: Window resizing should not cause unreasonable behavior in your simulation.

- **Exit**: When the 'q' or 'esc' key is hit, the program quits.
- **Multiple views**: The camera can be positioned in various locations, including one at a fixed location, one behind the boids and one to the side. See set-up description below.
- **Steerable Goal**: The boids will attempt to follow a steerable goal. If you do nothing, the goal flies at a constant velocity. You can slow it down and speed it up and turn it left or right, up and down through some keyboard/mouse combinations (for example, use the arrow keys for left/right/up/down and 'v'/'b' for slowing down and speeding up). The goal point should be rendered (although you might want to program a key to have the option of hiding it) and drawn in a manner that is clearly distinguishable from the boids.
- **README**: Write a detailed README that clearly outlines all the keyboard/mouse commands that are necessary for controlling your simulation. In addition, summarize your basic data structure and program set-up.
- **Replacing GL functions**: In preparation for switching to shader-based OPENGL, you should draw using vertex arrays. You not allowed to use `glBegin`, `glEnd` any more. In addition, you should implement functions that replace `gluLookAt` and `gluPerspective` (or `glFrustrum` if you use that instead), which computes your own projection matrix. Note that this replacement should be the last step, only attempted when your simulation works well with the given API calls.

Implementation Suggestions:

- **General Setup**: The ground lies along the x-y plane and the z axis points up to the sky. There should be a large ground terrain, which will provde your boids ample area to fly over. There is no requirement that the boids stay within this region, however. Near the center of the domain there should be an observation tower. Your boids may simply fly through it if you are not implementing obstacles for extra credit.

  For example, the ground can be a large square on the x-y plane of side length 20,000 units. It is rendered as an alternating 50x50 checkerboard pattern and thus each square is 400 units on a side. The boids have a speed of around 40 units per second and are roughly 10 units in length.

  Initial flock of boids can be placed randomly in a confined region of space. Say start with 10 boids placed randomly in a roughly 50x50x50 box near the point (2400, 150, 1200).
- **Misc**: It is highly recommended that you implement/use a vector library, to streamline the inevitably large amount of vector-based calculations. Since we don't have lights yet, rendering with wireframes will probably give better depth cue than flatly shaded polygons, or at least make sure every face has a different shade. The simplest bird object is modeled with 2 triangles joined along a common edge (a non-planar quad).
- **Goal Control**: The initial goal should be palced at a moderate distance away from the initial flock and should be given an initial velocity, so that the simulation starts in a reasonable state. (Say (2000, 0, 1000) with velocity (-40, 0, 0)). There are a few ways to handle the goal control.

You can implement the goal much like a boid (with a position P_g and a velocity vector $\vec{v}_g$), but it doesn't observe any of the boid behavior rules, obviously. Incremental speeding up and down can be handled by scaling the velocity vector with an appropriate factor. To move the goal up and down, add a vector to $\vec{v}_g$ that is parallel to the z-axis and whose magnitude depends on the magnitude of $\vec{v}_g$. Turning left and right is similiar, instead use a vector that is perpendicular to both $\vec{v}_g$ and z (use the cross product).

- **Multiple Views**: Your program should support at least three views which are based on the relative position of the flock and the goal. The user can switch between these via keyboard input.

  In the description below, let C denote the centroid of the flock, G denote the location of the goal. Let $\vec{u}$ denote the vector directed from C to G and let d denote the distance from C to G. Let $M = (C + G)/2$ be the midpoint between the flock centroid and the goal. Let r be the maximum distance of any boid to C, that is, it is the radius of the sphere centered at C that contains all boids.

  - **Default view**: 'v' key. This view is taken from an observer (a bird watcher?) located at a fixed position - the top of the observation tower, centerd over the origin. The view should be centered about M. You may adjust the filed-of-view to simulate a telephoto lens.
  - **Trailing view**: 't' key. This view is taken from behind and above the flock, and should include the boids and the goal position. For exmaple, one approach would be to mvoe backwards from C along the direction of $-\vec{u}$ to a distance of $d + 5r$ from C and then upwards (along the z) by a distance of $d + r$. Then take a view centered at M whose y field of view is 30 degrees.
  - **Side view**: 's' key. This view is taken from the right side of the vector $\vec{u}$ and above. For example, consider a vector $\vec{p}$ that is simultaneously perpendicular to $\vec{u}$ and to the z-axis. Starting from M we move in the direction of $\vec{p}$ by a distance of $d + 2r$. We then move up by a distance of $d + r$. Take a view centered at M whose x field of view is 40 degrees.

  Since these views are always perfect, it is useful to have controls that zoom in/out and up/down through keyboard input by modifying the distances used above.

Extra Credit Features:

- **Flapping**: Animate your boids so that they flap their wings. For the sake of realism, boids should not all flap at the same time.
- **Banking**: When a boid turns, it should bank in the direction of the turn. The angle at which the boid banks should depend on the sharpness of the turn.
- **Shadows**: Draw a shadow of each boid on the ground. This can simply be a vertical projection of the boid on the ground, but more realistic shadows of course earn more credit.

- **Obstacles**: Add some number of disjoint circular objects in the scene. The locations of the obstacles are specified in a reasonable way at run time (e.g. by giving their radii and center coordinates in an input file). Boids should avoid obstacles. When an obstacle is present, the boids are allowed to split up to avoid the obstacle. They should regroup in a natural way, once they have cleared the obstacle, assuming that the obstacle is not too large.
- **Multiple flocks**: Have multiple flocks of boids that avoid each other. You may assign new boids to flocks randomly.
- **Predator**: Implement a predator boid that fly around randomly (or under mouse control). The regular boids must avoid the predator at all costs, even if it means violating the flocking rules. Once the predator is at some distance, the boids resume regular flocking behavior.
- Feel free to add other features for extra credit.