Project G4

Due Date: April 2

Purpose

This project will allow you to code a well-known algorithm for simulating flocking behavior. In the process, you will use modeling ideas, three-dimensional transformations, perspective projection, and basic 3D objects.

Problem

Pixart is at it again, and is developing its next movie. Part of the plot revolves around a flock of birds constantly flying around near the main characters, causing lots of mayhem. They need to model flocking behavior, so they've hired you, for many thousands of dollars, to write a prototype program that will allow Pixart to better envision what the final movie might look like. The user will have several parameters to manipulate that will affect how the flock looks and moves. These options will be chosen by a fixed set in an OpenGL menu(s). All of this should be done in a three-dimensional, perspective space.

The algorithm for simulating flocking behavior was developed by Craig Reynolds, who presented his work *Flocks, Herds, and Schools: A Distributed Behavioral Model* at the 1987 SIGGRAPH conference (published in *SIGGRAPH Computer Graphics* and available from our library). You will use his algorithm to simulate your own "boids." A more straightforward algorithm is available at `http://www.kfish.org/boids/pseudocode.html`.

Input

The program should feature a menu(s) for the following features/parameters:

- Choose the size of the boid from several options.
- Choose an appropriate "closeness" parameter from a valid list.
- Choose an appropriate boundary movement parameter.
- Choose the number of boids to simulate from an acceptable range.
- Start/restart the animation with the chosen parameters or, if none were chosen, a default set of parameters.
- Quit.
- Other options are encouraged but not required. There are other parameters, such as wind speed, that can be added into the basic algorithm.

The user also has the following keyboard options:

- 4: Pause the animation.
- 6: Resume the animation.
- 7: Slow down the animation.
- 9: Speed up the animation.

Output

The program should produce a large window that the boids will fly around. The window represents a three-dimensional space. A boid should be a very simple (and small) representation of a 3D "bird." When the animation is started, the n boids (chosen by the user or the default value) should move about the screen area in the x, y, and z directions. Note that the x and y boundaries are obvious, but you must also program the z boundaries. Your program should follow Reynold's algorithm and exhibit appropriate flocking behavior, based on the input parameters. The boids should be oriented so they match the direction of travel. The boids should also get larger/smaller as they move in the z direction, which should be taken care of automatically by the perspective projection; be sure to make your view volume rather large, especially in the z direction, so that these size changes are obvious. Finally, there should be some sort of 3D "boundary" visible on the screen.

If the user desires to change the parameters, then the whole animation should be restarted. In other words, you do not have to make changes to the animation dynamically.

Specifics

- Use perspective projection. Some of the parameters could be user-defined, which might be interesting.
- The boids should be rendered in 3D. They can be custom-made by using your own defined vertices, or they can be made out of OpenGL pre-defined objects.
- Your default parameters should yield a nice, smooth animation that follows Reynold's algorithm. Changing the parameters to their extremes may yield slightly less smooth results, which is OK too.
- You are free to use any OpenGL functions. This includes transformation functions and display lists.

Notes

Once again, you should work on this project one portion at a time. I suggest the following steps:

1. Set up a 3D frustum in which your boids will move about.
2. Follow the algorithm so that basic boids flock appropriately on a 2D plane within your 3D space. That is, ignore the z value.
3. Make sure the various parameters, such as "closeness" and boundary conditions, work properly.
4. Create a better-looking boid.
5. Add in true 3D capability. By now, you should be very familiar with your code, and this shouldn't be too hard.
6. Add in other options in the order of your choosing.

Grading: Your grade is based mostly on how well your boids show the appropriate flocking behavior in a smooth animation in 3D. Animating good flocking behavior in 2D is worth 65 points. Modifying this to show the boids moving in 3D adds 15 points. Nicer boids are worth up to 10 points. Boids that point in the correct direction are worth 10 points. Note that all of these points are based on a smooth animation based on the given input parameters. Other options can give you some additional points; see me if you have ideas.

Send in your source code with your name, as usual. For example, I would send in gousieG4.cpp. If you have multiple files, bundle these with zip or tar.

All problems in computer graphics can be solved with a matrix inversion.
– Jim Blinn

There are about a dozen great computer graphics people and Jim Blinn is six of them.
– Ivan Sutherland (We will see one of Ivan's algorithms soon)