Boids

Members: Souren Papazian, Guanqi Luo, and Jae Hyun Kwon

Summary

Our project renders the movement and behavior of Boids. Boids is characterized as chaotic (splitting groups and wild behaviour), orderly, and unexpected behaviours, such as splitting flocks and reuniting. Our project provides realistic-looking representations of flocks of birds flying together in certain direction.

The main challenge was optimizing the algorithm of nearest neighbors and rendering 500 birds every step in an efficient way. For this we used octrees and display lists respectively.

All of the detail of the project is below. Each member discusses his part.

Implementation

Souren Papazian (ssp2155)

Files worked on:

Bird.java
Game.java
Octree.java
Sky.java
BoidUtil.java

Implementation:

1) Boids Behavior

The behavior of the boids was basically the heart of the project. The code for the behavior i inside BoidUtil.java. Basically boids have three rules they follow to act like a flock: cohesion, separation and alignment.

cohesion: change direction towards the average position of neighbors. separation: change direction to avoid crowding neighbors.
alignment: change direction towards the average direction of neighbors.

To implement this, first of all we needed a bird class which had a position, velocity and direction variables. All three are vectors. The bird also has a Move function which changed the position of the bird based on its current direction and velocity.

I wrote cohesion, separation and alignment inside BoidUtil. Basically given a bird and its neighbors, these functions change the direction of that bird slightly based on the characteristics of its neighbors.

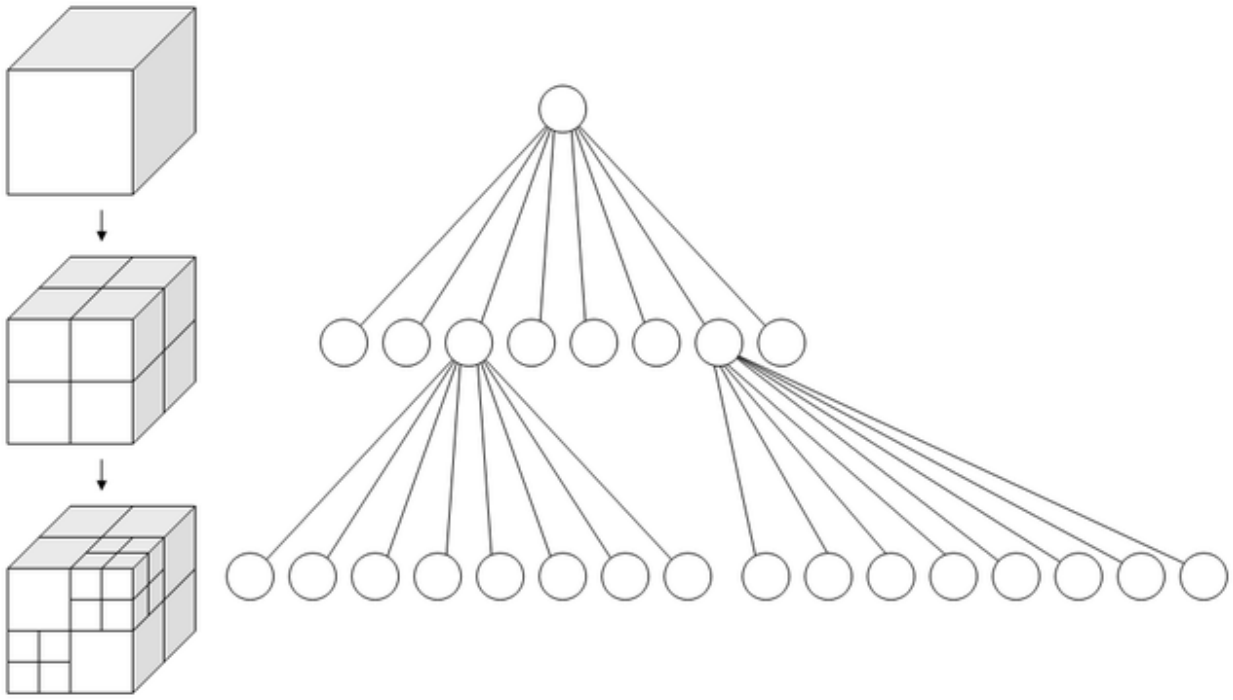
In the Game class (which is the main class), every step in the function updateLogic: for each bird that exists, the neighbors are fetched using the octree (I explain the octree in the next section) and then the three boid rules cohesion, separation and alignment are called to change the current birds direction. Each have a different range which I set to something that looks most natural. Cohesion checks the farthest range for neighbors, alignment ⅔ as far and separation ⅓ as far. I played around with these numbers a lot with some reasoning to make the flocking behavior look good.

2) Octree

An octree is a 3D spatial tree data structure. The logic behind is that when a certain area in it gets crowded with elements, it splits that area (cube) into 8 octants so that each octants only has a max of 5 elements in it. This way you have areas that are more crowded, split more and areas that have less elements split less. Here is the wikipedia of the octree for more info, but I'm sure you also know what it is: http://en.wikipedia.org/wiki/Octree

The octree basically makes a large amount of boids possible. The point of the octree for boids is to get all the boids in a certain range from a point very efficiently (basically the neighbors of a boid). If you look at the get_inRange function of the Octree.java class, it uses the stucuture to efficiently only check the distance between the elements around the element you are currently getting the nieghbors for. This actually took quite a while to perfect because it's not that simple of an algorithm. I found the algorithm here (pindyk slides) but this was for a quadtree and I had to adapt it to work for an octree (not so easy).

It turns out that it is most efficient to have max 5 elements inside each section of the octree. I measured it and the octree is exponentially faster than brute force nearest neighbors algorithm. In fact the brute force is what I implemented it with first and it was lagging really bad. Now it runs very smoothly.



3) Proper simulation

To actually get a proper simulation, I realized that you can't just move each boid every step and not worry about the other boids and the octree.

First of all, for it to be realistic, all the boids have to move at once. Meaning that if you loop over all the birds and move and affect them one by one, since they are dependant on one another (they need their neighbors info to act accordingly). This means that every step you need two copies of the boids, one which you use to compute each birds new direction and position and the other where you actually set this new direction and position. Then you delete the old one and render your new birds.

Second of all, every times the boids move, the octree structure changes. Meaning boids can change sections. I actually tried to remove and reinsert each boid one by one but that turned out to be slower than, deleting the whole old quadtree and reinserting into a new quadtree every step.

With these two things fixed, the simulation was realistic because the birds didn't take 'turns' moving, they all moved at once.

4) Boids Initialization

We initialize 500 boids in cube of size 800x800x800. They all have random locations and directions. They soon form flocks that fly around. We make them loop over every edge, meaning if they hit the right side of the cube, they spawn all the way on the left. It surprisingly doesn't lag with 500 birds, the octree optimization was definitely worth it!

Guanqi Luo (gl2483)

Files worked on:

Bird.java
Game.java
ShaderProgram.java
Body.java/LWing.java/RWing.java
Utilities.java
bird.vert/bird.frag/bird2.geom

Implementation:

1) Bird Model and animation

The bird model is created from a (.obj) mesh file. Then the model is divided into 3 parts (body, lwing and rwing) using blender. Each part is loaded to the corresponding classes using a DisplayList to speed up the process of creating the entire model. The flapping animation is achieved by applying simple forward kinematics to the wings so that rotation angle changes with time.

2) Shaders

There are 2 versions of the shaders, one with geometry shading and the other doesn't. Vertex shader is very simple and only setting vertices and normals. In fragment shader, I applied the phong shading model to the bird along with texture mapping from an image. I also applied Perlin noise with a high frequency to the shader so that it generates some pattern on top of the texture.
The goal of the geometry shader is to create the effect of feathers of bird. So for each triangle in the mesh, I created 3 extra triangles on top of the original and for each layer of triangles, two of the vertices stay the same as the original and one vertex move toward the normal direction so that the resulting triangle is tilted by a small amount. The mixing of the 3 layers results in the simulation of feathers.

Since geometry shader is only supported by GLSL3.3 and above, so I make the program so that it initially uses the shader without the geometry and user can switch to use the geometry shader by **pressing G** and switch back by **pressing O**. Note that geometry shader slows down the simulation quite a lot because it's computationally intensive.

without geometry

with geometry

Jae Hyun Kwon (jk3655)

Files worked on:

Game.java
Camera.java

Implementation of User Interaction:

How to play the game:

Press A: move left
Press S: move back
Press W: move forward
Press D: move right
Press Shift Key: move up
Press Space Key: move down
Mouse movement: camera rotation
Mouse Left Click: respawn a bird at the camera position if the camera is within the cage

1) First Person Camera Control
   Three important properties:
   position – A vector that will store the x, y and z co-ords of the camera.
   yaw – A float that will store the yaw (y axis rotation) of the camera.
   pitch – A float that will store the pitch (x axis rotation) of the camera.

We do not need to store the roll (z rotation) as a First person camera will not roll (tilt) but you could add this if you want your camera to tilt (some games use tilt to peek around corners).
With these properties, the program runs rotation based on mouse movement.

2) Respawning Birds

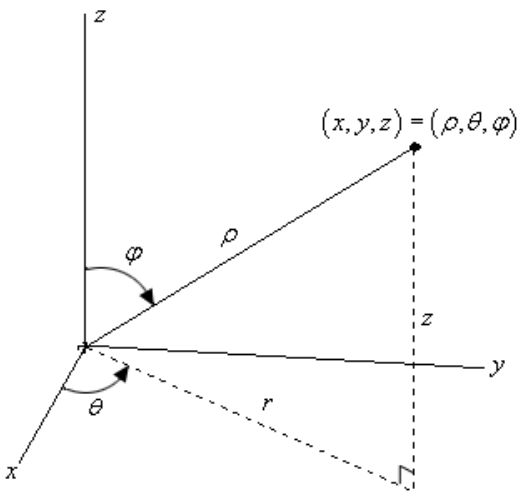
a. Respawn one bird per one click

b. Boundary

In order to set the boundary of the bird cage, I created a polygon boundary box and have a list of all the points and create a if statement with the conditions being set as equations to determine if the point is inside, which is a practice of linear programming. And then I check the points of one polygon to the condition of the conditions of collision. Example if you have a line with points 0,0 and 1,0 with a boundary box with points 0,0 0,1 1,1 and 1,0 then you would check each point with if(x < 1 & x > 0 & y > 0 & y < 1) where x and y are the point coordinates.

c. Direction

When you click the mouse, based on where camera is looking at, the program respawns a bird in the spherical coordinate direction with respect to the rotation angles of the camera.
Spherical coordinates consist of the following three quantities. First there is p. This is the distance from the origin to the point and we will require p>=0. Next there is theta. This is the same angle that we saw in polar/cylindrical coordinates. It is the angle between the positive x-axis and the line above denoted byr (which is also the same r as in polar/cylindrical coordinates). There are no restrictions on theta. Finally there is phi. This is the angle between the positive z-axis and the line from the origin to the point. We will require to be greater or equal to zero and less or equal to pi.
In summary, p is the distance from the origin to the point, phi is the angle that we need to rotate down from the positive z-axis to get to the point and theta is how much we need to rotate around the z-axis to get to the point. (Detail is shown below in the diagram.



Reference:

1. http://tutorial.math.lamar.edu/Classes/CalcIII/SphericalCoords.aspx
2. http://freespace.virgin.net/hugo.elias/models/m_perlin.htm
3. http://wiki.labomedia.org/images/1/10/Orange_Book_-_OpenGL_Shading_Language_2nd_Edition.pdf
4. http://www.informit.com/articles/article.aspx?p=2120983&seqNum=2
5. http://en.wikipedia.org/wiki/Octree
6. http://www.google.com/url?q=http%3A%2F%2Fdimacs.rutgers.edu%2FWorkshops%2FMiningTutorial%2Fpindyk-slides.ppt&sa=D&sntz=1&usg=AFQjCNElH5LAYSTqknu1kcSa9x2VggO-sg