Santiago Bonada, 100525252

Assignment 3: Red Tribe and Blue Tribe

Program Structure

VAO.cpp/hpp make a reappearance from my assignment 2 submission. They just hold the mesh for the ground model and monkey, as well as their shaders, loaded in the init() procedure in viewer.cpp. A class called boid is declared in boid.hpp with function definitions in boid.cpp. This class represents a single boid in the program – it holds position, velocity, acceleration, tribe, and the goal it has to reach. It has two constants, one that limits the range a boid searches around itself (FLOCK_RADIUS) and one that limits the magnitude of the acceleration (MAX_ACCELERATION). There is a static array holding the bounding box of the scene that boids cannot pass through.

There are two shaders, one for the monkeys (monkey.fs), and the other for the ground(ground.fs). The only difference is that the ground uses the normal vector as the base colour (it gives a nice definition to the flat faces of the obstacles). The fragment shader of both compute phong shading, the vertex shader is shared in general.vs.

So first thing that happens is the init() procedure (after OpenGL is set up). Here the meshes are loaded, then a grid gets initialized. The first state of the grid is all the spaces are denoted empty (the macro GRID_EMPTY), then the obstacles are placed in the grid (the macro GRID_OBSTACLE). GetGridCell(), which is in helper.cpp/hpp is used to convert opengl space to grid cells. After this the boids get initialized in the initBoids() procedure. A random number generator is used to randomly place the boids at different z locations; the x positions of boids in the same tribe are the same (they either start on one edge of the ground or the other). The number of boids created is based on the macro NUM_TRIBE. When boids get initialized, the grid is set where they are located with their index.

Once boids are initialized, the program begins. The draw function is displayFunc(), where each boid gets drawn. A for loop goes through the boids, where a model matrix is created for each that takes into account the position of the boid and it's orientation so that it faces the direction it is moving in. uniform variables are set here, such as the model matrix and the Eye location, as well as the colour which is set based on the tribe membership. After this for loop the ground get's drawn in much the same way. When this is all done the boids get updated in the updateBoids() procedure.

updateBoids() is mainly a for loop that iterates over very boid. First obstacle avoidance is computed by checking the grid cells near the boid within a for loop. A function that returns a boolean (infront() from helpers.cpp/hpp) determines if the direction of the boid intersects an area around the obstacle. If it does, or if the obstacle is close to the boid, an acceleration is applied to steer the boid away.

A second for loop handles boid-boid avoidance. Here we check the surrounding grid cells for nearby boids. If a boid is found in a grid cell we take the absolute dot product of their direction vectors; if this dot product is less than 1, then their directions intersect (thus meaning if they continue straight they will collide), and an acceleration is applied to steer the boid away from collision. In this for loop the positions and velocities are summed up for flock mates of a boid, which is then divided by the number of flock mates to get the centroid of the flock and the average velocity after the for loop.

Next an acceleration is applied to the boid if it's not currently directed towards the goal so that it steers towards the goal. The velocity is compared with the average velocity, and if it differs an appropriate acceleration in the boid's direction is applied so that it approaches the average velocity. The last acceleration applied is one that pushes the boid towards the flock's centroid.

Finally the timestep is executed, with a value of 0.033 seconds (a good point between animation speed and detailed movements). The timestep is done through the step() member function of boid(boid.cpp), which takes seconds as an argument. In this function, the acceleration is added to the velocity, the velocity is added to the position, then the acceleration decays. The timestep is used as a factor in all three calculations. In this function bounds checking also occurs. Finally, back in the updateBoids() procedure, the grid is updated so that it reflects the current state of the animation.

Techniques

Every time an acceleration is applied, it goes through the boid member function addAcceleration() which takes a vec3 as an argument. This function does some management, if the new acceleration after adding the acceleration has a large magnitude than MAX_ACCELERATION, the new acceleration gets capped to the maximum acceleration, but retains the same direction of the new acceleration.

For the obstacle avoidance acceleration, there are two accelerations applied. The first one is in a direction towards the side of the obstacle (dodge_dir) which is the boid direction minus the direction from boid to the obstacle. The second acceleration points directly away from the obstacle, but has a lower magnitude (a factor of 0.05).

For boid-boid avoidance acceleration, the vector that goes from the other boid to the current boid is used as the direction of acceleration, and it is multiplied by a weight inversely proportional to the distance between the boids as well as the radius of the boid as additional weight.

For keeping the boid moving to the goal, the dot product is taken between the direction to the goal and the direction of the boid. If the dot product is less than 0.5, then the direction of the boid is mostly not towards the goal and an acceleration is done in the direction of the goal. This acceleration is also applied if the boid is moving at a speed less than 1.0f units per second.

Material Required

Nothing that we haven’t used in the labs/previous assignments.